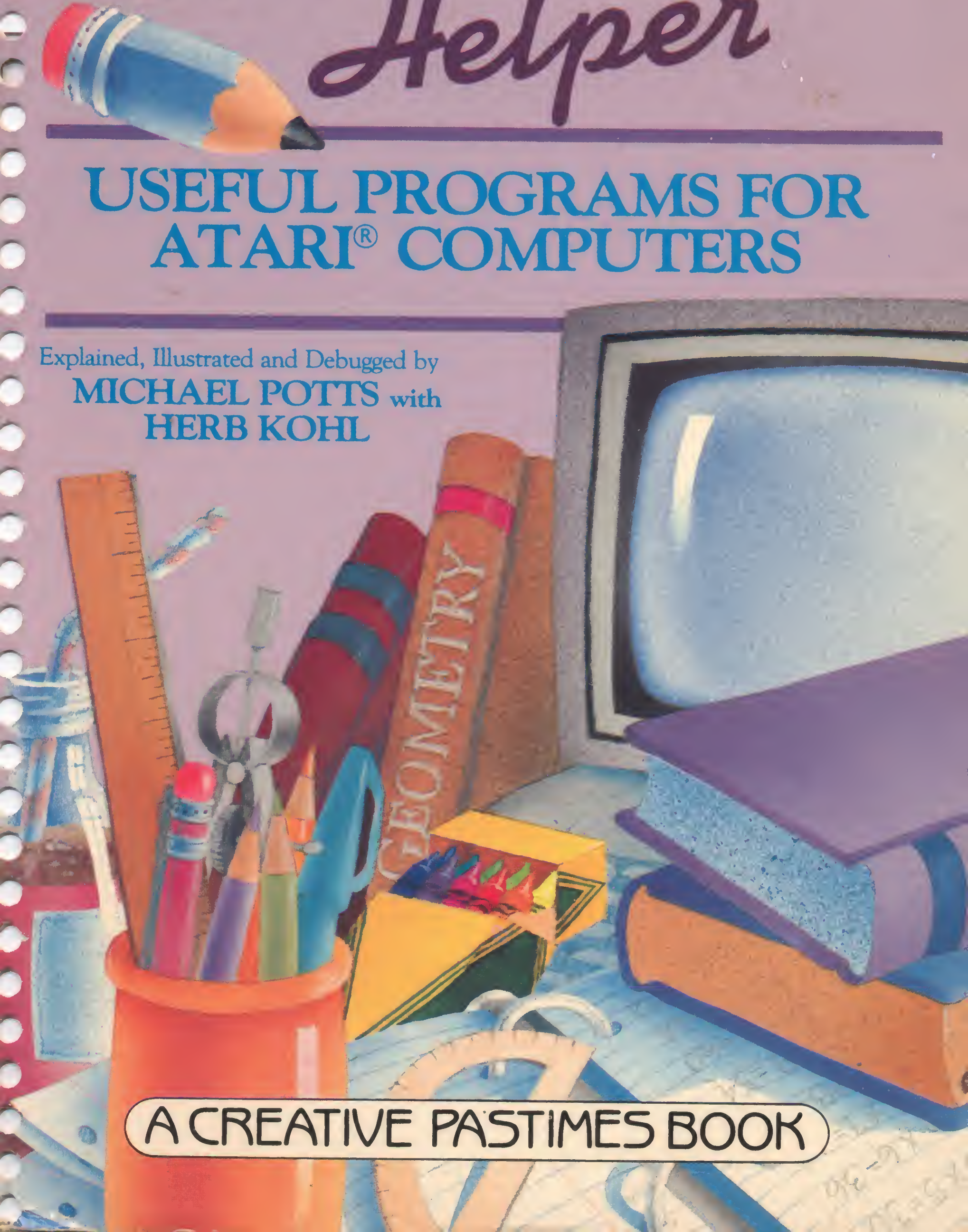
Homework Helper

USEFUL PROGRAMS FOR ATARI® COMPUTERS

Explained, Illustrated and Debugged by
MICHAEL POTTS with
HERB KOHL



A CREATIVE PASTIMES BOOK

Homework Helper

for the ATARI®

useful programs
for ATARI computers
explained, illustrated,
and debugged by

MICHAEL POTTS

with
HERBERT KOHL

A Creative Pastime Book
Reston Publishing Company, Inc.
Reston, Virginia
A Prentice-Hall Company

ISBN: 0-8359-2859-4

Interior design and production by Karen Winget.

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

Contents

acknowledgments

Thanks to Herb Kohl for his belief that this book could be done, and to Rochelle Elkan, without whose patient and thoughful collaboration this book could not have been done.

preface

This book uses the computer in an unusual way: as you build the programs in this book—computerized tools that will help you with school and work—you will be learning some of the tricks of the programmer's trade. When you are done, you will have a group of useful tools, *and* a good grasp on programming.

Interesting programs usually come already programmed: the real fun of working with a computer—making programs—has been grabbed by the packager, and you only get to use the program. In this book, you will learn about programming while building the tools for using the computer to learn about other things, too.

The book is designed for anyone who wants to learn programming, wants to provide students with useful tools, or both.

The four programs in the first section of the book are meant for younger students—they introduce the computer's keyboard and screen and help build symbol recognition and logic skills. But we can't expect young ones to program before they can read. An older person—"the programmer," maybe seven or older—will need to help. The first programs are carefully designed to help the novice over some of the humps all programmers encounter when they start, and to proceed with good programming habits from the very first.

The six programs in the second section are more advanced computer students; even so, programmers younger than age twelve will be able to write these programs, which will then be of genuine use in their studies. Some of these programs are more advanced in structure and concept, so as you build these programs, you gain more insight into the ways professional programmers develop working programs.

The last section contains six programs which will be of use to anyone who wishes to use the computer as a tool for organizing and simplifying daily life. And the writing of these programs will expose the programmer to some of the more advanced techniques available to a programmer working on the ATARI.

In this book you learn to use the computer while building useful tools—the computer is present in your learning as *subject,* as *means*, and *tool.* I have been working with micro-computers in classrooms, using that approach, since they first appeared there in the late 1970s. I have watched seven-year-olds use this approach to learn programming. For many students, the computer is as natural a part of the learning environment as the television and the textbook.

"Yes, but does it do anything useful?" is a question many home computer owners hear. With these programs on your computer, the answer is clearly Yes. There are programs in this book that anyone, even big people, can use for practical work. And again: learning the native language of a computer, in order to teach it to do your work, is the most useful learning of all.

I have written this book with a clear purpose always in mind: learning *and* computers are fun when embarked upon in the right spirit.

Michael Potts

a message to parents and teachers

With this book, you can turn an ATARI into a Homework Helper. The programs in the *first section* help pre-school and elementary students (age four to nine) practice elementary concepts—counting, symbols, and spelling. Very young children can play these games and learn with the computer. (The bigger person who helps type these programs in will learn a lot about programming in BASIC—more about that later.) The five intelligence games in the *second section* are for the use of older students—ages seven through thirteen—polishing these skills further, but there are learning challenges here for most adults as well. The study tools in the *advanced section* help you illustrate, organize, graph, and calculate.

This is not a read-only book: you should work with it and an ATARI computer. It doesn't matter if you've never touched a computer—or even a typewriter keyboard—before. On the way through this book, you learn to program in BASIC. When you are done with this book, you will have a well-trained silicon-based homework assistant, *and* you will know how to use BASIC to get computers to do what you want.

Most books that teach programming remind me of high school language textbooks: lots of word lists and verb conjugations, but no feel for the language or the people that speak it. In this book I try to interweave computer language with a plain-English explanation. Doing this book is the equivalent of living for two weeks with a family in the BASIC interpreter!

Many computers live in closets because owners find little useful to do with them. This book tackles that problem head on: people, students particularly, can use the programs in this book to go the extra distance needed to be an exceptional student.

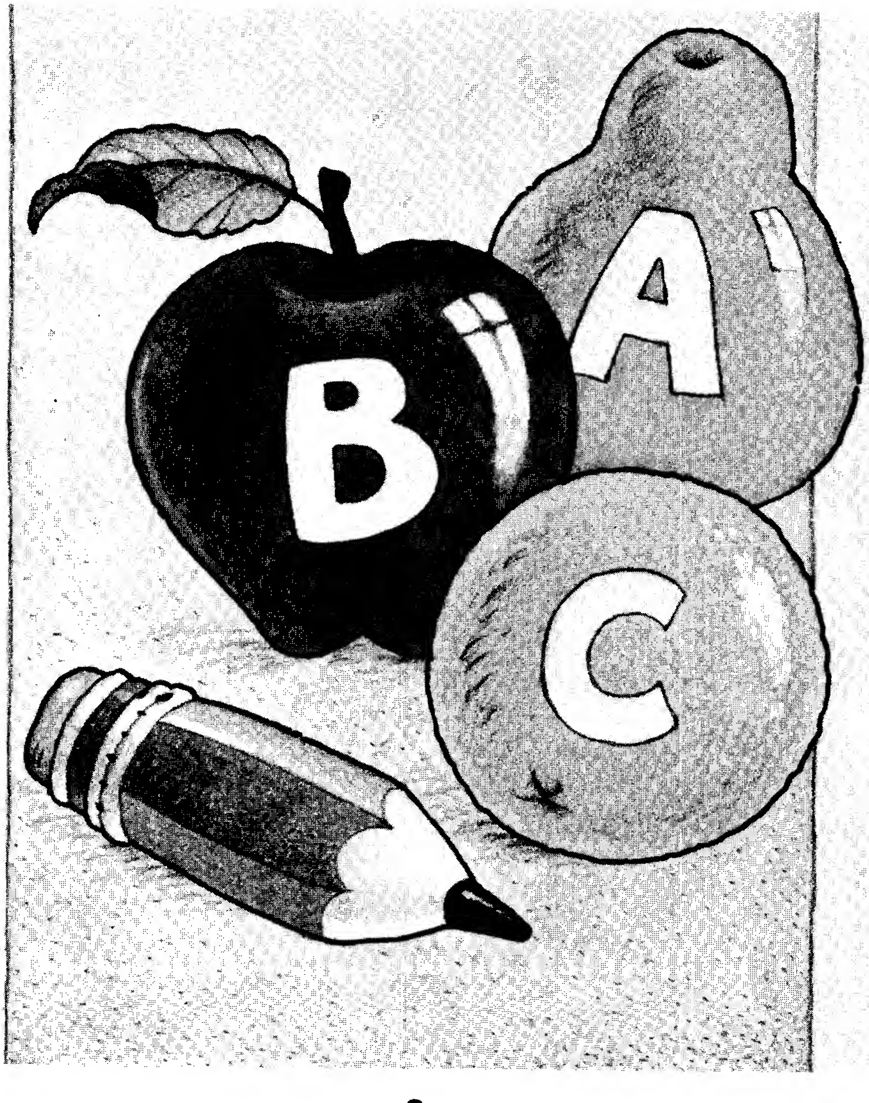
If you just bought an ATARI computer or are thinking about buying one to help your children with school, this book will build a useful partnership between your children and the computer.

If you bought (or are considering) an ATARI to teach yourself programming, this book will help you build the tools you will need to turn the computer into a working partner.

If you are a teacher with an ATARI in your classroom, this book should reside beside the computer as an invtiation to your students to make the computer do useful work. Take the computer and book home on weekends and write the programs; when you are done, you will be able to answer most of your student's (answerable) questions, and you will know enough to turn the computer into your classroom partner.

If you work with words and numbers, this book will help you turn your ATARI computer into a willing, skilled, tireless, accurate, and speedy assistant. Machines should work, and people should think—but to make machines work, you need to know how to ask. BASIC, the native language of most small computers, is easy to learn: even my youngest students have mastered it, working on tools like the ones in this book—tools to help with their work. This book makes computers a relevant and useful study.

early childhood games



Computers are like newborn children, who must be taught to crawl, speak, and walk—skills grown people take for granted. The four programs in the first section of this book may be more interesting for their programming techniques and structure than for what they do, unless you are fortunate enough to have a young learner around.

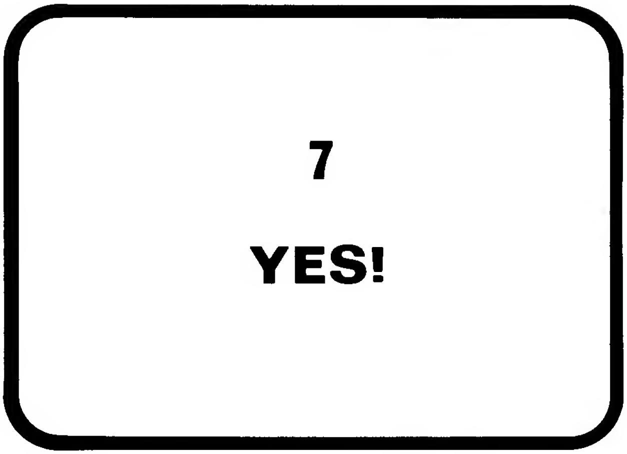
Learning with a computer is especially rich and rewarding for young children for a number of reasons. First, the current generation of home computers are well-suited to the learning tasks of the young—shape and symbol recognition, number games, and word-building. Second, computers represent a change as great as that made by telephones and television for earlier generations. Computers belong in the learning environment of the young, who will grow up to be completely at ease and in control of a technology that seems to many of their elders to be potent and sinister. Third, computers are growing up with our children: a child who masters an ATARI by age six—and that is quite possible—will be challenged by more capable, as yet unimaginable machines again and again as she grows up. Adults might view that challenge as unpleasant, but to children who "grow up" with computers, I expect it will be one of the most natural and enjoyable experiences in the world.

Learning with computers takes a number of forms; learning to *program* a computer is an experience much like learning a foreign language. To me, programming is negotiation with a Silicon Intelligence—the single-minded, literal, and logical presence within my machine. If I tell it to tie itself in knots or chase its tail, it does it unquestioningly and with lightning speed. Over the years, I have learned that even the simplest program benefits from my logical approach, or suffers when I am not sure what I want to do. Beginning programmers, especially younger ones, often bite off projects much larger than they can chew and find programming quite frustrating. I am reminded of George Washington Carver's story of his dialogue with the Almighty: "I want to know Everything about Everything," George said, and the answer came back, "How about *everything* about the *peanut*, George. It's more your size."

The programs in this section are easy to write, with just enough challenge for beginning programmers. They're about the right size.

NUMS

a symbol recognition game



```
7

YES!
```

DISCOVERING THE COMPUTER

NUMS is a very simple program, and even very simple programs have their challenge. Computers can be demanding partners. They are extremely literal-minded. Therefore, a good strategy for getting the job done is needed.

For this program to be useful as more than a programming project, you will need a young person (aged 2 to 5) to play the game when you have finished it.

The purpose of the game is to help build a quick, accurate association between the number keys on the computer and the symbols on the video screen. Many younger children find the keyboard, with all its symbols and keys, quite bewildering. Making friends with ten keys helps engage these young ones.

For the programmer, the purpose is to work with the computer with a job in mind and finish with a tool usable by a young child. The skills learned here will be useful when it comes to more complicated (and interesting) challenges in the future.

In NUMS, the computer randomly chooses a number (from an easily changed group defined within the program), and the player presses the corresponding key. When you get it, a colorful YES flashes on the screen.

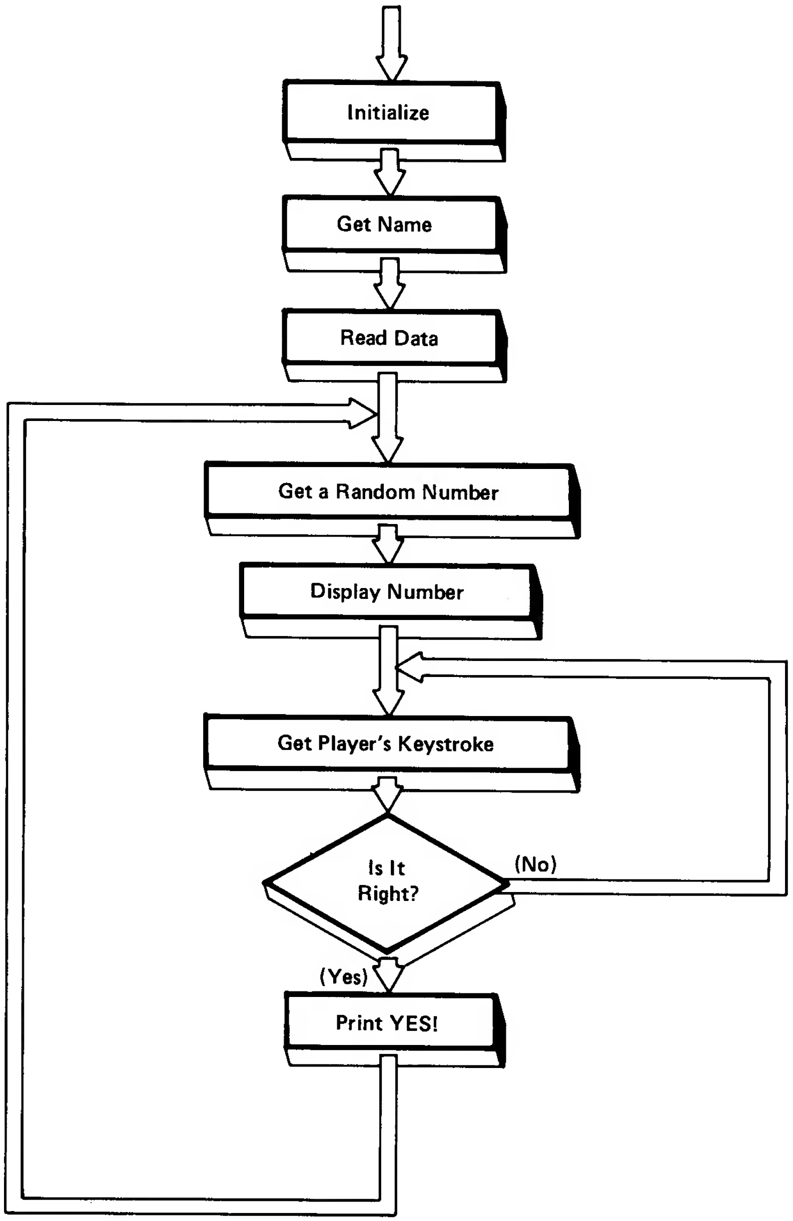
Younger children (aged 2-5) may be able to recognize only the numerals from one to five at first but will quickly learn the rest playing this game.

WRITING THE PROGRAM

Here's a flow chart and program listing for NUMS. If they make no sense to you, don't worry. There's a step-by-step explanation on page 8. Programming in BASIC, the native language of your ATARI, is easy, because you can build programs a few lines at a time and test them to make sure they work. We will use that system—professionals call it "modular programming."

Subroutines can be used from anywhere in a program or borrowed for several programs. In later programs you'll begin to recognize "old friends" among the program lines. You will learn to borrow them from earlier programs, so you won't have to type them in more than once. That's what computers are for: machines should work, people should think.

So let's get down to it, and start at the end!


Initialize
Get Name
Read Data
Get a Random Number
Display Number
Get Player's Keystroke
Is It Right?
(No)
(Yes)
Print YES!

The last line of the program is a one-line subroutine that makes up a random number R between a low number (LN) and a high number (HN). You will need this line in several programs, but you only need to type it once. If you build the program a piece at a time, and test each piece as you build it, you know for sure that your program works. If you type a program like this:

```
100 LN=0:HN=9
110 GOSUB 1300:PRINT R,
120 GOTO 110
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

and RUN it, your video screen will fill with randomly chosen numbers between zero and nine—unless you made a mistake. If it doesn't work, you only have four lines of code to debug. Lines 100, 110, and 120 are temporary (called a "stub" by programmers): they test the random number subroutine at line 1300 to ensure that it is working right.

ATARI 800 notes

Saving Your Work

It would be a good idea to practice saving your program while it's short. If you (or the computer) should happen to make mistake, you'd only have a few lines to type over again. If you have a tape system, make sure your recorder is ready, and type

```
SAVE "RANDOM"
```

If you have a disk system, type

```
SAVE"D:RANDOM
```

Dimensioning Strings

The ATARI is very picky about the way programmers use *string variables*—variables that contain text instead of numbers. An example is the N$—I call it N-string—in line 20 of NUMS. Soon it will contain the player's name. Line 2 tells the computer to save a place in memory 20 letters long for N$. If you forget, or I should say *when* you forget, to dimension a string, the ATARI will reward you with an ERROR 5 message.

THE BEGINNING OF NUMS

Let's plunge into the program. The first section is the *introduction:*

```
1 REM *************************** NUMS *
2 DIM A$(100),B$(3),N$(20),R$(1)
3 PRINT CHR$(125)
4 OPEN #1,4,0,"K:"
5 DATA 1,2,3,4,5,6,7,8,9,0,END

10 PRINT :PRINT "HELLO.":PRINT "MY NAM
E IS NUMS."
20 PRINT "WHAT IS YOUR NAME";:INPUT N$
:N=1
30 READ B$:IF B$="END" THEN 50
40 A$(N)=B$:N=N+1:GOTO 30
```

The lines before line 10 prepare (or *initialize*) the computer for the special task we have in mind for it. Line 2 sets up space for the string variables, line 3 clears the screen, and line 4 opens communications between the keyboard and the computer. Line 5 has the DATA that the computer will display on the screen.

Lines 10 and 20 introduce the program and find out the player's name. Lines 30 and 40 are a "loop," which reads the data in line 5 and stores it in an *array* called A$—testing every one as it goes to see if it's the END. Line 50 sets the low number and high number for the random number making subroutine. It is the last line in the set up part of the program.

ATARI notes

String Arrays

It would be nice to have a bunch of string variables to contain similar data—like the ten numbers for this program. ATARI MicroSoft BASIC lets you work with such an array of strings, but ATARI BASIC does not. We get around the problem by putting them all in one string (A$), which looks like this when it's been loaded up by lines 30 and 40:

```
    1234567890
```

That wouldn't be very useful, but fortunately we can ask for a part of a string at a time, like this:

You Type	*Computer Prints*
PRINT A$	1234567890
PRINT A$(1,2)	12
PRINT A$(5,10)	567890
PRINT A$(5,8)	5678
PRINT A$(7)	7890

In the last example, the computer provided everything from the number given er to the end of the string. With this substring-slicer, you can play many games with the computer's memory, as you will see as you read on.

THE MAIN PROGRAM

The main program pivots on line 50 like this:

```
50 LN=1:HN=N-1
60 GOSUB 1300:REM GETS A RANDOM NUMBER
70 GRAPHICS 2+16:POSITION 10,5:R$=A$(R
,R):PRINT #6;R$;
80 GOSUB 1000:B$=CHR$(K):REM GETS KEY
90 IF B$=R$ THEN 110
100 GOTO 80

200 FOR T=1 TO 800:NEXT T
290 GOTO 50
```

Line 60 sends out to the random number generating subroutine at line 1300. Line 70 pops us into GRAPHICS 2+16 and POSITIONs the cursor ten columns over and five lines down from HOME—the upper leftmost corner of the screen—and prints a randomly chosen string there.

Line 80 calls another subroutine: it sends program control to line 1000 to get a message from the keyboard and the outside world. Don't worry that there isn't any line 1000 yet. We'll write it soon. Until we do, the program won't run.

Skip lines 110-150 for now, but type in a temporary line like this:

```
    110 PRINT #6;"Y E S !"
```

Line 200 is an "empty loop." It doesn't do anything but count to 800, holding the display long enough for you to read it. Line 290 sends the computer back to the beginning of the main program loop at line 60.

Try it out: type RUN. If it doesn't work, find out why. Debugging is the fun part.

THE KEYBOARD SUBROUTINE

The three lines of code starting with line 1000 peek and poke around in the innermost recesses of the computer's memory to find out if any human has thumped on any key. This is the part where we tell the computer how to attend to us:

```
1000 I=PEEK(764):IF I=255 THEN 1000
1010 GET #1,K:POKE 764,255
1090 RETURN
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

Line 1000 PEEKs into the contents of memory address 764, where the computer maintains a constant flag about pressing of keys. If no key has been pressed since the last time we looked, we find the number 255 there. (If you are curious why 255, you will like ANYBASE on page 98.) If line 1000 discovers the 255 no-change message, then it looks again, and again, and again until someone pokes a key. Then (764) doesn't contain 255 anymore, and we can GET the most recently pressed key out of a buffer we OPENed back in line 4. Having received the keystroke, we POKE 255 back into (764) and RETURN to the line that sent us.

SOME FANCYWORK

When you have exterminated any bugs, you have time to dress up the program for use.

Now for more graphics.

```
110 POSITION 8,6:HN=5:LN=1
120 GOSUB 300:PRINT #6;CHR$(89+R*32);
130 GOSUB 300:PRINT #6;CHR$(69+R*32);
140 GOSUB 300:PRINT #6;CHR$(83+R*32);
150 PRINT #6;" !"

300 GOSUB 1300:IF R=2 THEN 300
310 IF R=3 THEN 300
390 RETURN
```

These lines play with the colors available for the letters in YES.

There is a copy of the whole program in the appendix so you can list and check your work. Writing programs is a very special kind of writing—more meticulous and exacting than any other kind of writing. If you misspell a word, the program misbehaves. But the payoff is that when the program runs right, you know it's well written.

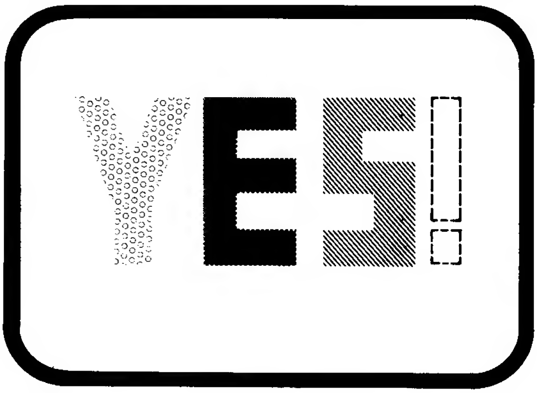
That is, if you saved it! What if the power went off now? This program is not quite finished, but good program practice says: SAVE IT anyway. If you have a tape system, type

```
SAVE "NUM1"
```

and follow the directions. If you have a disk system, type

```
SAVE "D:NUM1
```

Now if the power goes out, you're covered.



An editing trick: lines 130 and 140 are almost the same as line 120. You can LIST 120, then move the cursor up onto the listed line, change the 2 in the line number to a 3 (or 4), move your cursor to the right, and change the two numbers that differ, then press < RETURN >, and you've got the new lines at almost no cost to yourself. Of course, if that seems too complex, you can type the lines from scratch.

Now you get to debug the new parts—I had to fiddle with my colors to make them work the first time. Then BE SURE TO SAVE! It's a finished program now, so call it NUMS when you save it. Test your work, and when you're sure it's right, find someone to play it.

When players get tired of the number from zero to nine, you can change line 5's data. Here are examples.

The most-used keys:

```
    5 DATA E,T,O,A,I,N,R,S,END
```

The home row for touch typing:

```
    5 DATA A,S,D,F,J,K,L,;,END
```

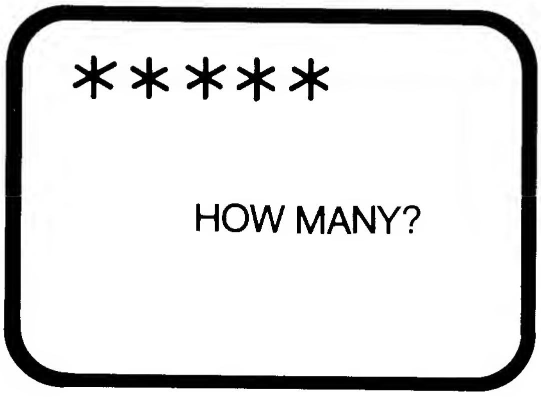
The beginning of the alphabet:

```
    3 DATA A,B,C,D,E,F,G,H,I
    4 DATA J,K,L,M,N,O,P,Q,R
    5 DATA S,T,U,V,W,X,Y,Z,END
```

You can use any keyboard symbols you want. Just make sure that END is the last entry in your DATA.

COUNTEM

a counting game

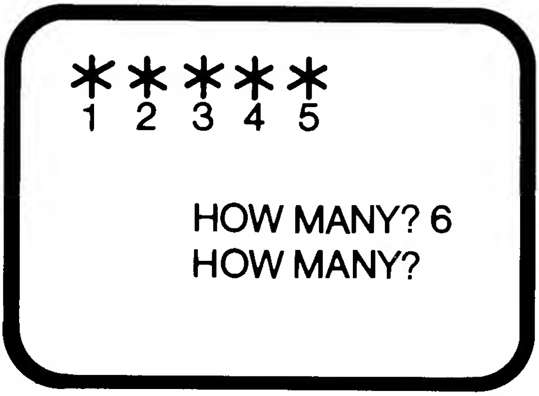


A COUNTING-SKILLS PROGRAM

This program was inspired by the Count, a TV character that delights in counting things—a delight shared by many young children. As soon as young players have found some of the keys they need on the computer, they are ready to strengthen associations between the keys and numbers.

For the programmer, COUNTEM is interesting because it introduces the concept of "borrowing from yourself," a strategy that saves a lot of work and confusion: if you can reuse a piece of a program (a module) that is known to work, the programming task is much easier.

What happens when the player gets a wrong answer? Computer programs that TEST ONLY say "WRONGO!" (and some even make an impolite noise!). I like programs that help the player LEARN. The computer, and the programmer, should be smart enough to offer help. Frequently, the help is nothing more than a quick reminder about how to count:



COUNTEM draws a random number of hearts on the screen, then invites the child to count them. (The programmer chooses the range of numbers to count.)

A simple switch, and the objects to be counted flash on the screen for only a moment; even adult players can challenge and build their pattern-recognition skills!

WRITING THE PROGRAM

This program uses some routines borrowed from NUMS. Wouldn't it be nice if you didn't have to type them in again? Well, you don't!

ATARI notes

Start out with NUMS in your computer's memory, and type

```
LIST 110-200
```

The computer will list the code for the colorful YES! block. Then type

```
LIST 300-1300
```

You'll see the rest of YES, and the keyboard and random number generator subroutines on your screen. Now type

```
NEW
```

and NUMS will be erased from the computer's memory, but not from the screen! To get the lines into your new program, use the cursor control keys to move the cursor into each line you want to borrow, and press {ENTER}.

for COUNTEM you need to borrow lines 110, 120, 130, 140, 150, 200, 300, 310, 390, 1000, 1010, 1090, and 1300. Now LIST your program. The trickiest parts are already done.

```
110 POSITION 8,6:HN=5:LN=1
120 GOSUB 300:PRINT #6;CHR$(89+R*32);
130 GOSUB 300:PRINT #6;CHR$(69+R*32);
140 GOSUB 300:PRINT #6;CHR$(83+R*32);
150 PRINT #6;" !"
200 FOR T=1 TO 800:NEXT T
300 GOSUB 1300:IF R=2 THEN 300
310 IF R=3 THEN 300
390 RETURN
1000 I=PEEK(764):IF I=255 THEN 1000
1090 RETURN
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

THE PROGRAM'S INTRODUCTION

```
1 REM ********************** COUNTEM *
2 DIM C$(1),D$(6),M$(12),Q$(1)
3 C$=CHR$(125):M$="            "
4 OPEN #1,4,0,"K:"
5 PRINT C$;M$;"* COUNTEM *"
10 PRINT D$;"WHAT IS THE LOWEST NUMBER
";:INPUT LL
20 PRINT "           THE HIGHEST NUMBER";:
INPUT HH
```

Line 3 sets up a "clear the screen" string, called C$, to erase old data and place the cursor near the screen's center. M$ is a margin, so that the writing isn't crowded against the left side of your screen. Line 5 uses both of these screen formatting strings to display the program's name.

Lines 10 and 20 get the lowest and highest numbers for the random number maker. When you play this program, remember to keep the numbers adjusted to the player. If you use numbers less than 1 or bigger than 9, the program will crash.

THE MAIN PROGRAM

```
30 LN=LL:HN=HH:GOSUB 1300:GRAPHICS 2+1
6:POSITION 1,5
40 FOR N=1 TO R:PRINT #6;" *";:NEXT N
60 POSITION 3,8:PRINT #6;"HOW MANY? ";
:GOSUB 1000
70 IF Q=R THEN 110
80 POSITION 2,4:FOR N=1 TO R:PRINT #6;
N;" ";:NEXT N
90 GOTO 60

290 GOTO 30
```

Line 30 sends out to our old friend the random number maker for a random number R, then prints R hearts on the screen. You make the * in line 40 with the key above the ATARI key. Lines 50 and 60 move the cursor down a bit and ask "HOW MANY?," then send the program to the keyboard subroutine at line 1000 for an answer.

There is some new code in the keyboard:

```
1010 GET #1,K:POKE 764,255
1020 Q$=CHR$(K)
1030 IF Q$<"1" OR Q$>"9" THEN 1000
1040 Q=VAL(Q$):PRINT #6;Q
```

Line 1030 checks to see if it's a legal key—one of the numbers between 1 and 9. If it's not legal, the computer ignores it and goes back to waiting for someone to press a key.

If it is a legal keystroke, the keyboard subroutine figures out the number value for the key, prints the symbol, and sends control back to line 60, the line that called the subroutine.

Line 70 checks to see if Q (the value of the key pressed) is the same as the number of asterisks. If it is, off to the colorful YES! box at line 140.

If not, line 80 moves the cursor back up beneath the hearts, and numbers them to help the player count. Line 90 moves the cursor back to the HOW MANY? then sends the computer back to line 60 for a new answer.

You need only one more line: line 290 sends the computer back to the beginning of the main program after the YES! display.

Time to test the program, then put it to use strengthening number concepts.

A challenge for people who know how to count: set the low number at 5 and the high number at 9, then try to estimate—counting is not fair—the number of hearts. After a little practice, you'll be amazed at how quick you get.

To speed things up a bit, change line 50 by adding a timing loop like this:

```
50 NEXT:FOR T=1 TO 50*R:NEXT T:GRAPHICS 2+16
```

If that goes by too fast, change the 50 to a larger number.

FINDME

coordinate geometry game

HIDE-AND-SEEK WITH A COMPUTER

Now that numbers are second nature, how about something more difficult? Two numbers! Coordinates are two numbers that define a location in a two-dimensional space, and this game plays with that idea. Ordered pairs—you always type the horizontal, or X-coordinate, first—are an important mathematical notion that many children never wholly grasp. This game is good practice for several classic games, including Battleship and Star Trek.

The main program for FINDME is very simple to write and debug. For more challenge, there is a second version that employs the computer's sound capabilities and demands a bit more of the programmer's and player's, skill.

```
                FIND ME!

Y
8  +  +  +  +  +  +  +  +  +  +
7  +  +  +  +  +  +  +  +  +  +
6  +  +  +  +  +  +  +  +  +  +
5  +  +  +  +  +  +  X  +  +  +
4  +  +  +  +  +  +  +  +  +  +
3  +  +  +  +  +  +  +  +  +  +
2  +  +  +  +  +  +  +  +  +  +
1  +  +  +  +  +  +  +  +  +  +
0  +  +  +  +  +  +  +  +  +  +
   0  1  2  3  4  5  6  7  8  9  X
CAN YOU FIND ME (X,Y)? _
```

```
                FIND ME!

   Y
   8  +  +  +  +  +  +  +  +  +  +
   7  +  +  +  +  +  +  +  +  +  +
   6  +  +  +  +  +  +  +  +  +  +
   5  +  +  +  +  +  +  X  +  +  +
   4  +  +  +  +  +  +  +  +  +  +
   3  +  +  +  +  +  +  +  +  +  +
   2  +  +  +  +  +  +  +  +  +  +
   1  +  +  +  +  +  +  +  +  +  +
   0  +  +  +  +  +  +  +  +  +  +
      0  1  2  3  4  5  6  7  8  9  X
 CAN YOU FIND ME (X,Y)?6,5
 YOU FIND ME
```

```
                FIND ME!

   Y
   8  +  +  +  +  +  +  +  +  +  +
   7  +  +  +  +  +  +  +  +  +  +
   6  +  +  +  +  +  O  +  +  +  +
   5  +  +  +  +  +  +  X  +  +  +
   4  +  +  +  +  +  +  +  +  +  +
   3  +  +  +  +  +  +  +  +  +  +
   2  +  +  +  +  +  +  +  +  +  +
   1  +  +  +  +  +  +  +  +  +  +
   0  +  +  +  +  +  +  +  +  +  +
      0  1  2  3  4  5  6  7  8  9  X
 CAN YOU FIND ME (X,Y)?5,6
 YOU MISSED ME!
```

FINDME makes up coordinates—pairs of numbers, like 2,3—and plots the corresponding point with an X on the screen. The player finds them.

A tricky variant, Sound FINDME, doesn't mark the point, but gives you musical clues to help you find it: the lower the note, the lower the number.

WRITING THE PROGRAM

```
1 REM ************************ FINDME *
2 DIM A$(1000),B$(20),D$(20),M$(12),C$
(12),R$(60),I$(1),T$(20),Q$(20)
5 C$=CHR$(125):M$=""
6 M$="":PRINT M$;"* FINDME *"
10 PRINT C$,"* FINDME *":PRINT "Y"
20 FOR R=8 TO 0 STEP -1:PRINT R;
30 FOR C=0 TO 9:PRINT " + ";:NEXT C:PR
INT :PRINT
40 NEXT R:FOR C=0 TO 9:PRINT "  ";C;:N
EXT C:PRINT "X"
50 HN=9:GOSUB 1300:X=R:HN=8:GOSUB 1300
:Y=R
60 VX=X:VY=Y:Z=88:GOSUB 1200
70 POSITION 1,22:PRINT "CAN YOU FIND M
E (X,Y)";
80 INPUT QX,QY:IF QX=X AND QY=Y THEN 1
00
90 GOTO 200
```

Lines 10 through 40 print the grid on the screen. Lines 50 and 60 locate the the Hider. Line 70 prints the words "CAN YOU FIND ME";the INPUT in line 80 supplies the "?",gets the player's answer, and sends the program to line 100 if the answer is right. If not, line 90 sends the program to line 200.

The subroutines are next:

```
1200 POSITION 4+(3*VX),2+(2*(8-VY))
1210 PRINT CHR$(Z);
1290 RETURN
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

Lines 1200-1290 calculate the screen position that corresponds to the Video-X and -Y coordinates (1200); print the appropriate character there (1210); and RETURN.

You recognize our old friend at line 1300—the random number maker. It has to work twice as hard in this game, because every answer needs two random numbers.

FINDME works now, up to the point where you give it a pair of numbers. If it doesn't, work on it until it does. Debugging is one of the primary skills of the programmer: you must learn to think linearly, like the machine. Exactly where does it stop doing the right thing? What was it supposed to do? What did it do? When you can answer that, list the code and exterminate the bug!

Now that the grid is drawn and the X plotted correctly, add the lines that check the player's guess.

```
100 PRINT "YOU FOUND ME!";
180 FOR T=0 TO 1999:NEXT T:GOTO 10
200 PRINT "  YOU MISSED ME";
210 VX=QX:VY=QY:Z=79:GOSUB 1200
220 POSITION 1,23:PRINT " ";:FOR T=0 T
O 500:NEXT T
230 PRINT "                          ";
240 GOTO 70
```

This is how the program works:

The main program begins at line 10 by clearing the screen and typing the game's name. Lines 20, 30, and 40 draw a grid on the screen. Line 50 gets random coordinates for the hider, and line 60 hides her.

Lines 70 and 80 ask the player to locate the hider, then check to see if the location is correct. If the player gets the right answer, lines 100 and 110 display "YOU FOUND ME!" for a decent period before clearing the screen and circling back to line 10 for another spot. If the player is wrong, line 90 sends program control to line 200.

Line 200 prints "YOU MISSED ME" and line 210 plots the spot the player entered for comparison. The most common error is giving the Y answer first; when the wrong point is plotted on the screen, it's easy to see the reversal. Line 220 leaves the message on the display for a moment, then line 230 wipes it out. Line 240 circles back to line 80 for another try.

ADVANCED PROJECT: SUPER-FINDME

In this program, the computer doesn't put an X to mark the hider's spot, but issues two pairs of tones to help the player find it. The tones are hints: the closer together the tones get, the closer you are the hider.

Super-FINDME blends the FINDME program with another program, or routine, called TWONOTE:

```
1900 FOR N=0 TO 9:READ NN:N(N)=NN:NEXT N
1910 N1=1:N2=8:REM *********** TWONOTE *
1920 GOSUB 2000:RETURN
2000 SOUND 0,N1,10,8
2010 FOR T=1 TO 40:NEXT T
2020 SOUND 0,N1,10,0
2030 FOR T=1 TO 4:NEXT T
2040 SOUND 0,N2,10,7
2050 FOR T=1 TO 80:NEXT T
2090 SOUND 0,N2,10,0:RETURN
2900 DATA 121,108,96,81,72,60,53,47,40,
53
```

The first lines (1900-1920) are called a "programming stub"—a sort of test-bed to make sure the program works right before it gets incorporated in a larger program. The notes are requested as variables Note1 and Note2. Be sure the numbers you supply for line 1910 are in the range from 0 to 9.

Line 2000 starts the sound generator, line 2010 keeps it on for an interval, and then line 2020 turns it off. The last number in the SOUND statement is the volume setting, and 0 is off. Line 2030 provides a bit of silence between the two notes, then lines 2040 through 2090 sound the second note.

When you are sure that TWONOTE is working (and saved in case something goes wrong), you are ready to merge the module with FINDME.

There are few changes necessary to make the sound module work. First, you need to add

```
:GOSUB 1900
```

to line 5, to initialize the sound generator. The subroutine for "displaying" the hider isn't at line 1200 anymore; we use the sound subroutine at line 2100, so line 60 needs to change, too.

Borrowing lines

To merge TWONOTE into FINDME to make Super-FINDME, make sure TWONOTE is in the computer's memory by typing

```
    LIST
```

Now type

```
    NEW
```

and then load FINDME from disk or tape. When the computer says READY, move your cursor to the top of the TWONOTE listing on your screen—you should see the cursor block flashing on the 2 in line 1900 of the listing. Now press {RETURN} once in every line of TWONOTE remaining on the screen—that should be ten times.

If you list your program now, you should find it is ten lines longer—the sound module from TWONOTE has been appended to FINDME.

```
1 REM ************************ FINDME *
2 DIM N(10),C$(1),M$(12)
5 C$=CHR$(125):GOSUB 1900

60 N1=N(X):N2=N(Y):GOSUB 2000
70 POSITION 1,22:PRINT "CAN YOU FIND M
E (X,Y)";
80 INPUT QX,QY:IF QX=X AND QY=Y THEN 1
00
90 GOTO 200
```

There is some added code between lines 100 and 180, the "Correct" Fanfare:

```
100 PRINT "YOU FOUND ME!";
110 N1=N(X):N2=N1:GOSUB 2000
120 N1=N(Y):N2=N1:GOSUB 2000
130 N1=N(3):N2=N(5):GOSUB 2000:GOSUB 2
000
140 N1=N(7):N2=N(8):GOSUB 2000
180 FOR T=0 TO 499:NEXT T
```

and similar changes between 200 and 290:

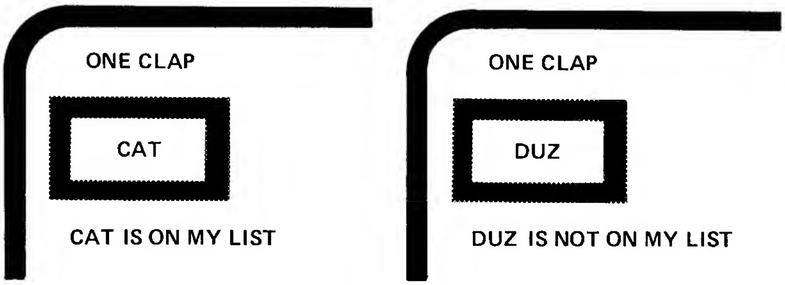
```
200 PRINT "  YOU MISSED ME";
210 N1=N(QX):N2=N(X):GOSUB 2000
220 N1=N(QY):N2=N(Y):GOSUB 2000
270 POSITION 1,23:PRINT " ";
280 PRINT "                          ";
290 GOTO 70
```

You can take out the subroutine lines from 1200 to 1290; it isn't used anymore.

The new program should run now. Test it. There is a full listing of SFINDME in the appendix for troubleshooting purposes.

An interesting footnote: you have written a program (SFINDME) that works very nicely without the video screen. (Of course, you can't turn the TV off, because the computer relies on it for the sounds as well as the picture.) Would it be possible to build a computer that didn't use the video at all? How would a computer for blind people work?

ONECLAP
a word-discovery game



"Is this a word, Mommy?" asks the young child, just learning to build words out of sounds. Exploring the way letters combine into words is an exciting quest for children aged five through seven. But English is an illogical language, and many words that should be spelled one way, *duz* for example, are spelled in quite another, *does*. It takes a lot of memory to know the spellings of the most misspelled words. But that's no reason why your computer can't know and help a young child discover a smaller class of words. At this age, children are fascinated by truly simple words like CAT, PET, and SIT: a letter or two, then a vowel (A, E, I, O or U) and a final consonant. I call these One-Clap words, because when you sing or say them, clapping your hands as you go, these words get only one clap, where other words get more: computer, for example, gets three claps.

This program is an ambitious programming project for the beginning programmer. It is nothing particularly complicated, but a number of modules must cooperate to do the program's work. Good luck! (Accurate typing helps, too.)

ONECLAP is really two games in one program. In the first game, the player presses any alphabet key, and the computer shows the letter inside the box on the screen. When the player presses < RETURN >, the computer replaces the letter with a short wod containing that letter. The computer searches for a matching word randomly, so pressing the same letter several times will give the player several words containing the chosen letter. Playing with the computer in this way, the player learns the words in the computer's vocabulary.

When the player is ready to move on, typing any two (or more) letters sends the computer to the second game (or "mode" as they say in the computer world). The child types in a simple word, and the computer compares it to its vocabulary. If the computer finds the word, it flashes the YES! blcok; if not, it confesses that the word "is not on my list" and makes a note of it for future reference by someone with a bigger vocabulary.

Children quickly learn the "one clap" concept by clapping along with the sounds of their speaking or singing. A "one-clap" word has only one syllable, but that's a hard term to explain to younger players. This program uses an even narrower definition: *a one-clap word has only one syllable, no fewer than three nor more than four letters, and its second-to-the-last letter is A, E, I, O, or U.* It turns out that there are hundreds of such words—see how many you can find—and they comprise a large majority of most children's first written words. This program only supplies a few, but children delight in finding more. If the programmer maintains the program by adding newfound "good words" after each session, the program becomes a fascinating mirror of the player's burgeoning written vocabulary.

WRITING THE PROGRAM

Several new concepts make their appearance in this program. We can build the program one module at a time, and the whole thing shouldn't take more than an evening.

The first lines prepare the computer to work with the special kind of program we are writing. DIM A$(4000) creates space for an array of memory locations to contain the program's vocabulary.

```
1 REM ************************ ONECLAP *
2 DIM N$(20),MO$(10),A$(4000),B$(1000)
,T$(40),F$(4),W$(4),I$(1),O$(1),V$(5),
C$(25)
3 DIM P(1000),W(5)
4 OPEN #1,4,0,"K:"
7 V$="AEIOU":RT=300
10 C$=CHR$(125):C$(2)="             *
ONECLAP *":PRINT C$:MO$="ANY KEY":MO=0
20 PRINT :PRINT "WHAT IS YOUR NAME";:I
NPUT N$:GOTO 3000
30 GOSUB 300:IF LEN(T$)>1 THEN 55
40 GOSUB 400:G=G+1:IF G<10 THEN 30
```

Lines 3000-3200 read the vocabulary from the data lines 9000-9990 into the array called A$(V,W), where V is the vowel number (starting with 0 for A), and W is the word number. After the last word for each vowel is stored, the computer keeps track of how many words it has for each vowel in an array of numbers called W(V). This module is designed to read as many as 101 words for each vowel from the data, provided, of course, that the A words come before the E words, and the very last data entry is ZZZ.

```
3000 PRINT "EXCUSE ME,";N$
3010 PRINT "I AM READING MY LIST NOW."
3020 PRINT :W=0:LL=0:PP=0:T$=V$(1,1):V
=0:W(0)=0:P(0)=0
3030 READ W$:LW=LEN(W$):IF W$(LW-1,LW-
1)>T$ THEN 3100
3040 PRINT W$;" ";:LL=LL+LW+1:IF LL>33
 THEN PRINT :LL=0
3050 A$(PP+1)=W$:PP=PP+LW:W=W+1:P(W)=P
P
3090 GOTO 3030
3100 V=V+1:W(V)=W:IF V=5 THEN 30
3110 T$=V$(V+1,V+1):GOTO 3030
9000 DATA BAT,CAT,EAT,FAT,HAT,MAT,PAT,
RAT,SAT,TAT,VAT,WAX
9010 DATA BET,GET,JET,LET,MET,NET,PET,
SET,VET,YET
```

```
9020 DATA BIT,FIT,HIT,KIT,PIT,SIT,TIT,
WIT,ZIP,QUIT
9030 DATA COT,DOT,GOT,HOT,JOT,LOT,NOT,
POT,ROT,SOT,TOT
9040 DATA BUT,CUT,GUT,HUT,JUT,NUT,PUT,
RUT
9990 DATA ZZZ
```

Line 3020 gets everything set for reading the data. Line 3030 READs a word—it must be a string, because it is text, not numbers—counts the number of letters in the word, and checks to see if the second-to-the-last letter is the same as last time. If not, it shoots off to line 3100 to keep track of the number of words for the current vowel. If it is the same vowel, the program moves on to line 3040 where the word is printed, the line length on the display is tallied, and if it is greater than 33 a new line is started. Line 3050 adds the new word to the vocabulary list stored in A$, notes the ending position of the new word, adds 1 to the word count, and goes back to line 3030 for the next word.

You can RUN the program now to test your work so far. It should read its vocabulary, then encounter an error at line 30—it tries to find line 300, which isn't written yet. If you type

```
    PRINT A$
```

after the crash, you should get something like this:

```
    BATCATEATFATHATMATPATRATSATTATVATWAXBET . . .
```

The next logical place to work is on lines 300-390, so the program RUNs farther.

```
300 GRAPHICS 2+16:PRINT #6;MO$
310 POSITION 8,5:PRINT #6;"?";:POSITIO
N 8,5
320 T$="":T=0:GOSUB 1000
390 RETURN
```

Line 300 shifts the screen display to a different GRAPHICs mode (which clears the screen), prints the mode, positions a ? near center-screen, repositions the cursor so the next keystroke will wipe out the ? and then sends out to line 1000 for a keystroke. We need our keyboard subroutine—do we have to type it in all over again? No!

Borrowing lines from other programs

It's smart to let the computer do the work for you. If you already have a working keyboard module—you do if you typed in NUMS or COUNTEM—you can avoid mistyping and debugging by following this procedure:

SAVE your work. (It is time you did that anyway. If someone kicked out the plug, you'd be sorry!) Then type

```
    NEW
```

and

```
    LOAD"D:NUMS
```

for a disk-storage system. If you are using a tape system, check your ATARI manual for loading instructions.

LIST line 4 and lines 1000-1300, then type NEW again. ReLOAD the incomplete version of ONECLAP you saved at the beginning of the last paragraph, and move your cursor up into the lines you listed from NUMS. Press {RETURN} in each of the listed lines; as you do that, the line is entered into your computer's program memory exactly as it shows on the screen.

Zip! the keyboard subroutine has been installed in your new program at very little cost and with a high degree of reliability.

```
1000 I=PEEK(764)
1010 T=T+1:IF T<RT THEN 1050
1020 IF MO>0 OR LEN(T$)=0 THEN 1050
1030 POSITION 6,10:PRINT #6;"PRESS [RE
TURN]";:T=0
1040 POSITION 8+LEN(T$),5
1050 IF I=255 THEN 1000
1060 GET #1,K:POKE 764,255
1070 IF K=155 THEN RETURN
1080 I$=CHR$(K):PRINT #6;I$;:T$(LEN(T$
)+1)=I$
1090 GOTO 1000
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

There are some necessary additions to the keyboard subroutine beginning at line 1000. If you're wondering what is special about

155 (in line 1070, it's ATARI BASIC's internal code for the {RETURN} key; line 1070, for instance, translates, "if the input key is not equal to {RETURN} then print it and add it to the text string."

You can test this module directly after you have it typed: type

```
GOSUB 1000
```

Nothing exciting happens—the computer just sits there and hums. Type something in, like

```
HELLO, COMPUTER
```

and press {RETURN}. The computer responds with READY. Nothing very interesting? Ask the computer to print T$—you can take a short-cut and type

```
? T$
```

The computer should respond by typing your message again: the keyboard subroutine displayed your keystrokes, while capturing them in the test string.

Line 1300 is our old friend the random number maker, borrowed already from NUMS.

Lines 400-490 match single keystrokes from the keyboard with words containing those letters (from the first part of the game).

```
400 F$=T$:GOSUB 500:IF F<0 THEN 420
410 HN=W(F+1)-W(F):GOSUB 1300:WP=W(F)+
R:W$=A$(P(WP)+1,P(WP+1)):GOTO 480
420 HN=4:GOSUB 1300:W=0:NW=W(R+1)-W(R)
425 NW=W(R+1)-W(R)
430 WP=W(R)+W:W$=A$(P(WP)+1,P(WP+1)):I
F T$=W$(1,1) THEN 480
440 IF T$=W$(LEN(W$)) THEN 480
450 W=W+1:IF W<NW THEN 430
460 R=R+1:W=0:IF R<5 THEN 425
470 R=0:GOTO 425
480 POSITION 8,5:PRINT #6;W$
490 GOSUB 800:RETURN
800 FOR T=0 TO 750:NEXT T:RETURN
```

The complicated expression in line 410,

```
W$=A$(P(WP)+1,P(WP+1))
```

extracts a single word from the vocabulary string. The storage strategy works like this:

```
P( )   0   1   2   W(1)   W(1)+1        W(5)

A$       BATCAT ... WAXGET       ...     RUT
```

Example: if we want the second word, its starting position is one character to the right of the end of the previous word—P(WP)+1, or position 4. It ends at P(WP+1)—position 6. (The WP counter starts at 0, and lags one behind the word count, so WP=1 for the second word.) A$(4,6)="CAT".

The lines from 500 to 590 set up a variable F to work with the vowels in the words.

```
500 F=-1:IF F$="A" THEN F=0
510 IF F$="E" THEN F=1
520 IF F$="I" THEN F=2
530 IF F$="O" THEN F=3
540 IF F$="U" THEN F=4
590 RETURN
```

With these modules typed in and debugged, the first part of ONECLAP should work—and the hardest programming is done. To finish the program you need:

```
50 MO$="ONE CLAP":MO=1:GOSUB 300
55 LT=LEN(T$):IF LT<3 THEN 250
60 F$=T$(LT-1,LT-1):GOSUB 500:IF F=-1
THEN 250
70 W=0:NW=W(F+1)-W(F)
80 WP=W(F)+W:W$=A$(P(WP)+1,P(WP+1)):IF
 T$=W$ THEN 100
85 W=W+1:IF W<NW THEN 80
90 GOTO 250
100 POSITION 3,6:PRINT #6;"IS ON MY LI
ST.":GOSUB 800
110 NR=NR+1:IF T$="QUIT" THEN 700
190 GOTO 50
```

Line 200 and further take care of words not on the list.

```
25Ø POSITION 1,8:PRINT #6;"IS NOT ON M
Y LIST.":GOSUB 8ØØ
26Ø B$(LEN(B$)+1)=" ":B$(LEN(B$)+1)=T$
29Ø GOTO 5Ø
```

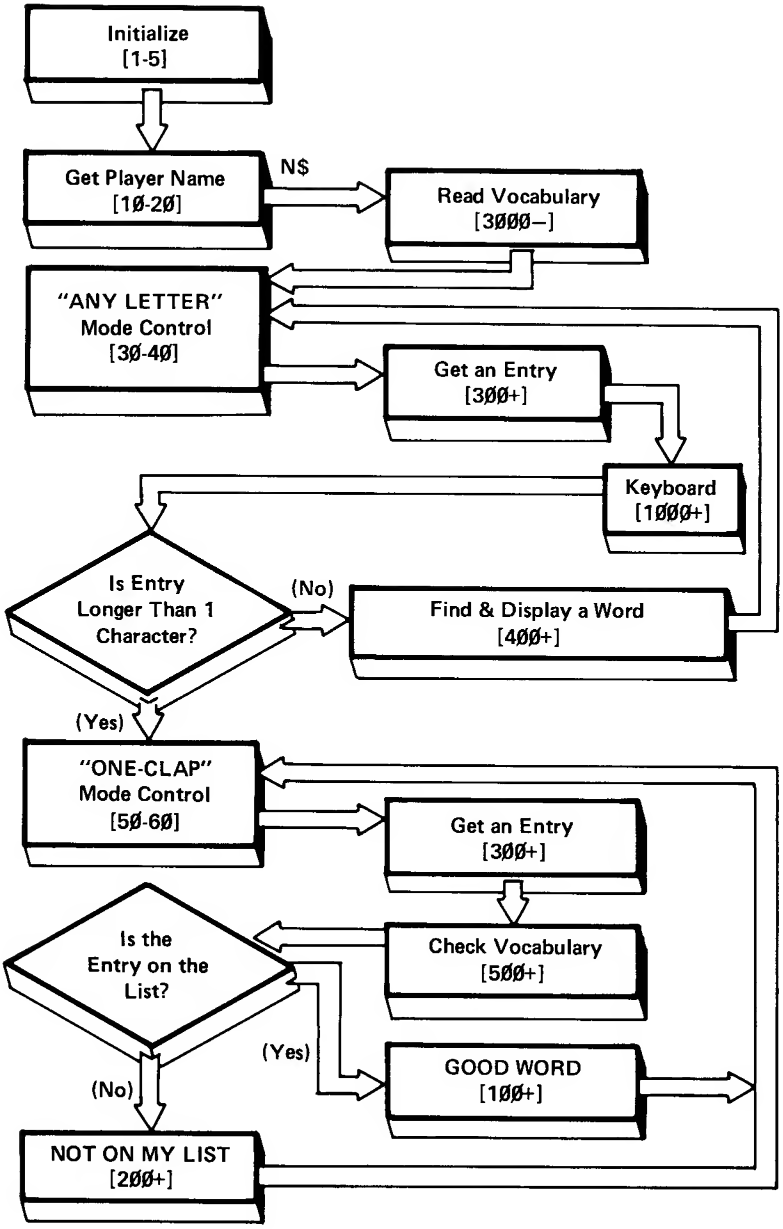
Long program! But now it's ready to test. On the next page you will find a programming tool, called a flow chart, to help you debug this program. Follow the lines on the chart while the computer follows the program logic. If the computer does something unexpected, you'll know where to look for a problem. Computers are literal to a fault: if a single character is wrong, the computer will either complain or do the wrong thing.

Once you have ONECLAP performing the way it should, you can add lines 700-900, which let the player QUIT the game and see the words the computer doesn't have on its list.

```
7ØØ PRINT C$:PRINT :PRINT
71Ø PRINT N$;" FOUND ";NR:PRINT ,"GOOD
 WORDS."
72Ø PRINT :PRINT "WORDS NOT ON MY LIST
:":PRINT :P=1:LB=LEN(B$)

73Ø PP=37:IF LB-P<37 THEN 79Ø
74Ø IF B$(P+PP,P+PP)=" " THEN 76Ø
75Ø PP=PP-1:GOTO 74Ø
76Ø PRINT B$(P,P+PP-1):P=P+PP+1:GOTO 7
3Ø
79Ø PRINT B$(P,LB):END
```

There is a full listing of ONECLAP in the appendix.


Initialize
[1-5]
Get Player Name
[1Ø-2Ø]
N$
Read Vocabulary
[3ØØØ–]
"ANY LETTER"
Mode Control
[3Ø-4Ø]
Get an Entry
[3ØØ+]
Keyboard
[1ØØØ+]
Is Entry
Longer Than 1
Character?
(No)
Find & Display a Word
[4ØØ+]
(Yes)
"ONE-CLAP"
Mode Control
[5Ø-6Ø]
Get an Entry
[3ØØ+]
Check Vocabulary
[5ØØ+]
Is the
Entry on the
List?
(Yes)
GOOD WORD
[1ØØ+]
(No)
NOT ON MY LIST
[2ØØ+]

school skills

The next six programs take the computer from the crawling and walking stage to the running and jumping stage. There should be something interesting and useful for anyone old enough to press the keys. There is one game that will have the quickest of mathematicians scribbling away with pencil and paper along with the rest of us: it's possible, but tough.

The computer is a smart blackboard: write a "little program" that teaches some specific lesson quickly. these six programs polish various school skills—logic memory, spelling, math, history, and word problems—as painlessly as possible.

The programming involves several new techniques. As you construct these programs, you will see the reasons for building certain parts first, like building the roof and walls to keep the rain off, but only after the foundation and floor are solid.

PATTERNS

8 15 16 11 ?

```
         PATTE  S
        PATTE N
       PATTE NS
      PATTER
     PATTER S
    PATTERN
```

PATTERNS-a pattern recognition game

This program makes up mathematical puzzles like

```
1 3 ? 7 9    and    8 15 16 11 ?
(series one)        (series two)
```

The player's job: decipher the rule and supply the unique missing number. Such puzzles often find their way into intelligence tests and math textbooks; the ability to deduce the answers quickly is often taken for intelligence.

Some patterns, of course, are easier than others: the first series is the odd numbers, and the missing member is 5—a six-year-old can easily find that. The second one is not quite so easy—the answer is zero.

```
18   36   54   ?
WHAT IS MISSING?
```

$$\begin{array}{ccccccccc}
1 & & 3 & & ? & & 7 & & 9 \\
& \underbrace{\qquad}_{2} & & \underbrace{\qquad}_{2} & & \underbrace{\qquad}_{2} & & \underbrace{\qquad}_{2} &
\end{array}$$

$$\begin{array}{ccccccccc}
8 & & 15 & & 16 & & 11 & & ? \\
& \underbrace{\qquad}_{7} & & \underbrace{\qquad}_{1} & & \underbrace{\qquad}_{-5} & & \underbrace{\qquad}_{-11} & \\
& & \underbrace{\qquad}_{-6} & & \underbrace{\qquad}_{-6} & & \underbrace{\qquad}_{-6} & &
\end{array}$$

There is a systematic method for solving all but the alphabetic puzzles with pencil and paper. Write out the series, then bracket each pair of numbers and note the difference between them. Then bracket each pair of differences and again calculate the differences. Repeat the process until you recognize the pattern, and you'll be able to work backwards to find the missing number. Good luck!

PATTERN is the computer version of an age-old number game: it presents a series of numbers with some mathematical rule governing their relationship, and the player tries to find the rule and supply the missing number. This program uses five different *algorithms* (fancy computer talk for rules) to generate the sequences: additive, multiplicative, incremental, exponential, and alphabetic, to give them properly mathematical names. Mixing them up randomly, this game can sharpen the number skills of almost anyone. Does practice make perfect?

The program begins by asking you to select a difficulty level from 0 (easy) to 9 (hard). The complexity of each series is based on this level. The computer presents a series with one missing member, which you calculate and type in. If you are right, the computer makes up another series, but if you're wrong, it shows you the same series with a different missing number.

WRITING THE PROGRAM

PATTERNS lends itself to modular construction, because it involves several different methods for deriving number series—each method gets its own set of lines. Start with a framework that includes two different series-generation algorithms, make sure they work, and then add more modules.

The program uses our old standby, line 1300: the random number maker. You can copy it from another program following the procedure on page 16. The main framework of the program goes like this:

Line 5 sets the low number variable LN for the random number maker. Line 60 gets the difficulty level; line 70 calculates the high number for randomly choosing the series algorithm, which is then chosen in line 80.

```
1 REM ********************** PATTERNS *
2 DIM Q$(3),P$(11):P$="PATTERNS    "
5 LN=1
60 PRINT "PATTERNS":PRINT :PRINT :PRIN
T "EASY (0) OR HARD (9)";:INPUT DF
70 PRINT :HN=INT(DF/2)+1:IF HN>5 THEN
HN=5
80 GOSUB 1300:ON R GOTO 100,200,300,40
0,500
```

Lines 1000-1090 are a very simple "keyboard" to get the player's answer. Lines 1100-1190 leave lots of space for polishing up the scoring routine—see the suggestions at the end of this section.

I believe we have seen line 1300 before.

```
1000 PRINT
1010 PRINT "WHAT IS MISSING";:INPUT Q$
1020 IF Q$="" THEN 1090
1030 IF ASC(Q$)>64 THEN 1090
1040 Q=VAL(Q$)
1090 RETURN
```

```
1100 PRINT "YOU ARE RIGHT."
1190 GOTO 70
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

The rest of the program makes up and prints the series. The easiest module is additive, and it serves as a pattern for "knocking off" the other algorithms.

```
100 HN=4*(DF+1):GOSUB 1300:S=R
120 HN=2*(DF+1):GOSUB 1300:I=R
130 HN=2:GOSUB 1300:L=3+R
140 HN=L:GOSUB 1300:M=R
150 FOR N=1 TO L
160 IF N=M THEN PRINT " ? ";:GOTO 180
170 PRINT " ";S+I*(N-1);" ";
180 NEXT N:A=S+I*(M-1):GOSUB 1000:IF Q
=A THEN 1100
190 GOTO 140
```

Line 100 sends out for a random starting number for the series; 120 sends for an increment (the amount to increase each time a new series number is calculated). Line 130 gets the length of the series, and line 140 decrees which of the series members will be missing. The lines from 150 to 180 are a loop that checks to see if this member is missing or printed (160), calculates and prints the latter170),ndloops baif the series has not yet reached its allotted length—that's the FOR (line 150) . . . NEXT (180) loop. When you understand what a loop does, you have perceived the power of the computer.

$$\mathbf{S}\underbrace{\qquad\quad}\mathbf{S+I}\underbrace{\qquad\quad}\mathbf{S+I^{*}2}\ldots\mathbf{S+I^{*}N}$$

ATARI notes

It's easy to duplicate lines in ATARI BASIC. All you do is LIST the original lines, move the cursor onto the line you want to duplicate, change its line number (and anything else you need to change to make it exactly right). Is everything exactly right? Excellent. The last thing you do before leaving the line is press {RETURN}. That snap-shoots another line into the computer's program memory.

You can test-drive the program as it stands (but be careful, you'll crash if you try any difficulty over 2!) but I wanted something a little more interesting, so I added another module: the multiplicative:

```
200 HN=4*SQR(DF+1):GOSUB 1300:S=R
220 HN=2*(DF+1):GOSUB 1300:I=R
230 HN=2:GOSUB 1300:L=3+R
240 HN=L:GOSUB 1300:M=R
250 FOR N=1 TO L
260 IF N=M THEN PRINT " ? ";:GOTO 280
270 PRINT " ";S*I*N;" ";
280 NEXT N:A=S*I*M:GOSUB 1000:IF Q=A T
HEN 1100
290 GOTO 240
```

If you have the feeling you have seen those lines before, you are remembering the module we just finished. You won't be surprised by the next two modules, either.

```
300 HN=DF:GOSUB 1300:S=R
320 HN=DF+1:GOSUB 1300:I=R
325 HN=INT(DF/2):GOSUB 1300:I2=R
330 HN=2:GOSUB 1300:L=3+R
340 HN=L:GOSUB 1300:M=R
350 I1=I:FOR N=1 TO L
360 IF N=M THEN PRINT " ? ";:A=S+I1*(N
-1):GOTO 380
370 PRINT " ";S+I1*(N-1);" ";
380 I1=I1+I2:NEXT N:GOSUB 1000:IF Q=A
THEN 1100
390 GOTO 340
```

This algorithm has two increments (and gets particularly nasty if you let negative numbers in: that's what those mean-looking lines at 325, 326, and 327 accomplish. When you get this module working, play around with the numbers and relationships of this section (if you're interested in becoming a mathematician).

And now for something utterly different: letters! Words are a kind of series, and the logic is certainly different, especially in

English. The module beginning at line 400 shifts the program from the cooly numeric to the jungle of textual calculation: strings.

```
400 IF FR=Ø THEN 16ØØ
41Ø HN=FR:GOSUB 13ØØ:R=R-1:A$=D$(P(R)+
1,P(R+1)):P=1
42Ø L=LEN(A$):HN=L:GOSUB 13ØØ:M=R
43Ø HN=2:GOSUB 13ØØ:I=R:IF I=1 THEN 45
Ø
44Ø I=-1:P=L
```

```
45Ø IF P=M THEN PRINT " ? ";:Z$=A$(P,P
):GOTO 47Ø
46Ø PRINT " ";A$(P,P);" ";
47Ø P=P+I:IF P=Ø OR P>L THEN 49Ø
48Ø GOTO 45Ø
49Ø GOSUB 1ØØØ:IF Q$=Z$ THEN 11ØØ
495 IF I<Ø THEN 44Ø
496 P=1:GOTO 45Ø
```

The first line sends to line 1600 only the first time this module gets called—those lines are next. The next three lines are familiar. Line 440 sets things up to step backwards through a word—like this: S D R A W C A B. The computer language we are using, BASIC, has a very powerful way of playing with words. We'll see more of that later. If the string-chopping in line 410 doesn't make much sense, please check the explanation of word-storage on page 29.

```
16ØØ DIM D$(1ØØØ),A$(2Ø),Z$(1),P(1ØØ):
N=Ø
161Ø READ A$:IF A$="ZZZ" THEN 169Ø
162Ø LD=LEN(D$):P(W)=LD:D$(LD+1)=A$:W=
W+1:GOTO 161Ø
169Ø P(W)=LEN(D$):FR=W:GOTO 41Ø
9ØØØ DATA BIRTHDAY,WASHINGTON,FISH,COM
PUTER,HORSE,BOAT
999Ø DATA ZZZ
```

There are dozens more possible algorithms that could be included in this game—and lots of memory in the computer, too—but I propose to give you only one more: exponents.

```
500 HN=SQR(DF):GOSUB 1300:S=R
520 HN=2:GOSUB 1300:I=R+1
530 HN=2:GOSUB 1300:L=4+R
540 HN=L:GOSUB 1300:M=R
550 FOR N=1 TO L:Z=INT(S*N^I+0.5)
560 IF N=M THEN PRINT " ? ";:A=Z:GOTO
580
570 PRINT " ";Z;" ";
580 NEXT N:GOSUB 1000:IF Q=A THEN 1100
590 GOTO 540
```

(The world of numbers is so much cleaner to program in!)

A bit of fancywork to dress up the beginning of the program, and we can wrap this one and move along to another kind of patterning.

```
10 GRAPHICS 2+16:P=1:X=1:Y=1:POKE 764,
255:R=1
20 POSITION X,Y:PRINT #6;P$(P,P);:K=PE
EK(764):IF K<>255 THEN GRAPHICS 0:GOTO
 60
30 P=P+1:IF P>8+R THEN P=1
40 X=X+1:IF X<19 THEN 20
50 X=1:Y=Y+1:IF Y<10 THEN 20
55 Y=1:POSITION 1,5:PRINT #6;"< PRESS
 ANY KEY >":HN=4:GOSUB 1300:R=R-1:GOTO
 20
```

This short module plays with GRAPHICS 2+16 to make patterns on the screen. If you leave this display running long enough for the color-cycling called ATARI Attract Mode to begin, you have quite an interesting display.

There are many more interesting relationships and interrelationships for making this kind of puzzle, and finding and defining them is a superior kind of intelligence-building game.

REMEMBER
polish memory skill

7 N Ø Q R 1 C

CAN YOU TYPE WHAT YOU SEE?

Another trait frequently identified as intelligence is the ability to glimpse random strings of symbols, like

```
7 8 3 5 2            or            7 : A ) Q # 2
```

then reproduce them: sort of rote-memory muscle-building. Of course some people will be able to handle much longer strings than others. In writing computer aided instruction, the concept of a program's "ceiling" defines the point past which learners cannot benefit from a program. This program is a good example of a game without a ceiling: the lower levels of difficulty present short strings for a long interval, but the highest levels are much too long and fast for me! In other words, this game should challenge anyone.

REMEMBER times and scores the play—interesting computer functions for any kind of gaming program. The methods and modules that time and score can be easily transported to other programs.

REMEMBER devises strings of numbers, letters, and typographical symbols, flashes them on the screen for a measured period, then clears the screen and invites the player to reproduce the string. Playing difficulty ranges from trios of numbers shown for several seconds (level 0) up to nine characters displayed for less than a second (level 9). You must pay constant, focussed attention to the screen, and your typing must be error-free.

The program is written "open-ended" so that you can add an elaborate timing and scoring section.

WRITING THE PROGRAM

The keyboard and random number maker appears once again in this program; you might like to borrow it from a program already in memory.

```
2 DIM C$(13),T$(10),Q$(10),M$(8),R$(1)
,I$(1)
4 OPEN #1,4,0,"K:"
5 C$=CHR$(125):C$(2)="REMEMBER...  ":PR
INT C$
6 M$=""
9 TK=0.015:SK=33
10 LN=0:PRINT :PRINT :PRINT
20 PRINT "0=EASY 9=HARD  ...HOW TOUGH"
;:INPUT DF:TD=200*(11-DF)
30 L=3+(DF/3):HN=9:IF DF>5 THEN HN=36
35 IF DF>7 THEN HN=22
40 T$="":FOR N=1 TO L:GOSUB 1300
50 IF R>9 THEN 70
60 R$=STR$(R):GOTO 80
70 R$=CHR$(54+R)
80 T$(LEN(T$)+1)=R$:NEXT N
```

Lines 5-30 get the computer ready for work: getting the player's difficulty level and calculating the time delay (20), the length of the strings to be devised, and setting the high number in accordance with the difficulty level. The FOR . . . NEXT loop from line 40 to 80 puts the test string together—line 60 adds a space and a number produced by STR$(R) each time around, line 70 takes care of letters and symbols.

The rest of REMEMBER manages the video display:

```
100 PRINT C$;DF;M$;:FOR N=1 TO LEN(T$)
:PRINT " ";T$(N,N);:NEXT N
110 FOR T=1 TO TD:NEXT T
120 PRINT C$;DF;M$;"?";:Q$="":T=0:POKE
 764,255
130 GOSUB 1000:T=T+1
170 IF K=155 THEN PRINT :PRINT :GOTO 2
10
180 Q$(LEN(Q$)+1)=I$:PRINT I$;" ";
190 IF Q$<>T$ THEN 130
200 PRINT :PRINT :PRINT "EXACTLY!"
210 IF Q$="" THEN 300


290 FOR T=1 TO TD:NEXT T:GOTO 40
```

Lines 1000-1090 are the keyboard routine that captures the player's keystrokes and compares them with the computer's string after each one. As soon as the player has completed entering the symbols—assuming they are exactly right—the program passes into the scoring process. If the player isn't precise, a {RETURN} submits the incorrect string for evaluation. The delay at line 290 maintains the rhythm of the program; it is exactly as long as the viewing cycle in line 110.

```
1000 I=PEEK(764):T=T+1:IF I=255 THEN 1
000
1010 GET #1,K:POKE 764,255
1020 I$=CHR$(K)
1090 RETURN
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

Once the program is performing properly, you can add a scoring routine:

```
220 LT=LEN(T$):LQ=LEN(Q$):SC=0:FOR N=1
 TO LT
230 IF N>LQ THEN 250
```

```
240 IF T$(N,N)=Q$(N,N) THEN SC=SC+1
250 NEXT N:PRINT "SYMBOLS RIGHT:";SC,
255 PRINT INT(100*SC/LT);"%"
260 SC=INT((SK*L)/((T*TK)*(SC/LT)))+1
280 PRINT "TIME: ";T*TK;" SECONDS":PRI
NT "SCORE ";SC:TS=TS+SC:NQ=NQ+1:TT=TT+
T*TK

300 PRINT C$;DF:PRINT
310 PRINT "SCORES:"
320 PRINT "TOTAL SCORE           ";TS
330 PRINT "STRINGS ATTEMPTED ";NQ
340 PRINT "AVERAGE SCORE/$     ";TS/NQ
350 PRINT "TOTAL SECONDS        ";TT
360 PRINT "AVERAGE TIME/$      ";TT/NQ
390 FOR T=1 TO 3*TD:NEXT T:GOTO 40
```

Lines 210 to 250 count the number of correct symbols-in-position. Line 260 calculates a "SCore" based on time and the correctness of the entry, and line 280 displays the results and keeps track of the score.

To see the scoreboard, press {RETURN} only (lines 300-390).

ADVANCED TOPICS

The timer (lines 120 and 130) is tricky because its count is governed by the number of computer cycles inside the timing loop—in this program, that's how long it takes the computer to do all the steps between lines 130 and 190. The difficulty is compounded by the fact that different ATARI computers seem to take different amounts of time to accomplish these steps. The time constant TK (in line 9) may have to be calibrated for the computer's "seconds" to correspond to "real time." The easiest way to do this is by carefully clocking the time between the appearance of the ? on the video screen—when the clock starts—and your last keystroke. Adjust TK until the computer time is accurate.

An adventurous programmer could adapt the scoring module to several of the other programs in this book. Recalibration of the clock would be required.

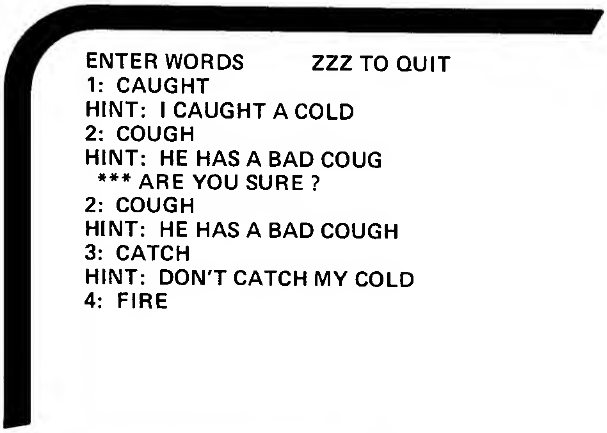
SPELL
a spelling driller

Spelling, at least in English, is not very different from remembering random strings of letters. There are two ways the computer helps. One way is with spelling-drillers, like the program in this chapter, that present words-to-be-learned in an interesting and instructive way.

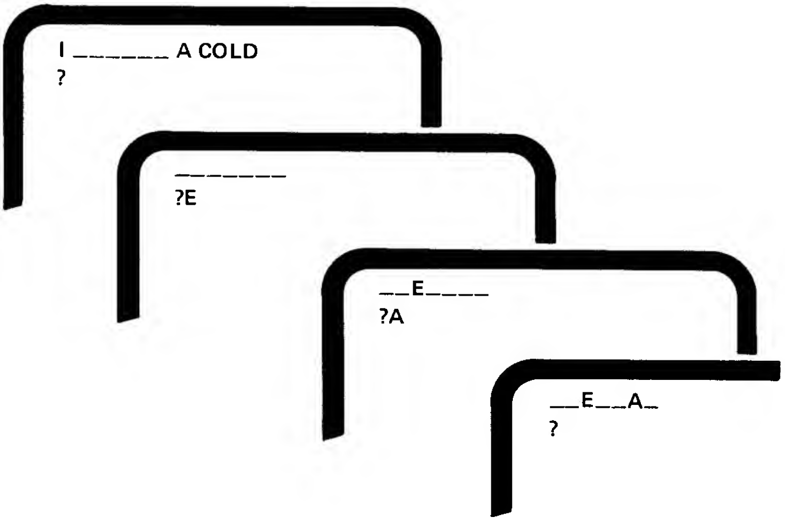
Another way is through spelling checkers, programs that comb through a written work for unrecognizable words, then offer the writer a chance to survey the dictionary and, if necessary, correct the spelling. [This book was checked for mistakes using a program called ProofReader. See the comments about Word Processing in the appendix.] Is it possible that the computer will liberate future children from spelling drillers?

Everybody learns in a slightly different way. This program uses four different ways to try to find everyone's good side: first, you type your list (and, if desired, a set of "hints"); next, you play REMEMBER (from the last chapter); then you supply the correct spelling for each hint; finally, you play Hang-Man with the spelling list.

For the programmer, this program is an easy challenge, because several different modes must work together within a single program.



```
ENTER WORDS        ZZZ TO QUIT
1: CAUGHT
HINT: I CAUGHT A COLD
2: COUGH
HINT: HE HAS A BAD COUG
 *** ARE YOU SURE ?
2: COUGH
HINT: HE HAS A BAD COUGH
3: CATCH
HINT: DON'T CATCH MY COLD
4: FIRE
```



```
I _______ A COLD
?
```

```
_______
?E
```

```
__E____
?A
```

```
__E__A_
?
```

SPELL uses four different practice strategies to help learn spelling. The first method is the straightforward entry of the spelling list. After entering the words, you are asked for hints for each of the words on the list—a short sentence or phrase for each word. (You can skip the hint-entry by pressing {RETURN} instead of a hint for the first word.)

```
CATCH   PLAY ----- WITH ME.
CAUGHT   I ------ A COLD.
```

When all the words are entered, SPELL goes into the "Flash" mode; it flashes the word on the screen for a moment, then clears the screen, and you type the word in. If you are right, you are informed; if not, the correct spelling is shown. When you have gotten ten words right, the program promotes you.

The "Clue" mode shows the clue, and you are invited to provide the missing word. Again, if you get it right, the computer tells you, but if you miss, the computer shows the right word. When you have gotten twenty more words right you go on.

The "Hang-Man" mode is familiar to all: the computer presents you with the right number of blanks, and you press the letters you think belong in the word. If you are right, the computer puts the letter in the right place, or places, if the letter occurs more than once in the word; but if you're wrong, the computer keeps track. You only get twice as many wrong guesses as there are letters in the word, so proceed thoughtfully.

WRITING THE PROGRAM

SPELL can be put together a mode at a time. The input section and the word flasher look like this:

```
1 REM *************************** SPELL *
2 DIM A$(1000),B$(20),D$(20),M$(12),C$
(12),R$(60),I$(1),T$(20),Q$(20),H$(38)
3 DIM P(3),Q(3),W(50),X(50),Y(25)
4 OPEN #1,4,0,"K:"
5 C$=CHR$(125):FOR N=2 TO 12:C$(N)="":
NEXT N:PRINT C$
6 M$="":PRINT M$;"* SPELL *"
8 P=1:FOR N=1 TO 3:READ T$:R$(P)=T$:P(
N)=P:P=LEN(R$):Q(N)=P:P=P+1:NEXT N
9 TD=1000:LN=1:NW=1:PW=1:H=0
10 GOSUB 500:REM GET DATA
20 FOR N=1 TO ND:PRINT A$(W(N),X(N))
30 IF H=0 THEN PRINT A$(X(N)+1,Y(N)):P
RINT
40 NEXT N
45 H=1:GOTO 200
```

```
50 FOR T=1 TO TD:NEXT T
60 HN=ND:GOSUB 1300:PRINT C$;M$;A$(W(R
),X(R))
70 FOR T=1 TO TD:NEXT T:PRINT C$;M$;:I
NPUT Q$
80 IF Q$=A$(W(R),X(R)) THEN 100
90 PRINT "SORRY, IT'S ";A$(W(R),X(R)):
WR=WR+1:GOTO 130
100 HN=3:GOSUB 1300:PRINT :PRINT R$(P(
R),Q(R))
110 RI=RI+1
120 FOR T=0 TO TD:NEXT T:IF H=1 THEN 2
00
130 IF RI<10 THEN 60
```

The code in line 8 reads the "YES" messages that flash when you get it right; line 9 sets the time delay, the lowest number (for our old reliable random number maker), and the length of the data. Please adjust any of these (except LN) to please yourself. There's an unattached wrong-answer counter in line 90. Could you hang anything on it?

You need two more routines to make the program work. The lines starting at 500 allow the player to enter data for study. The keyboard and random number maker are familiar (and could be borrowed from another program.)

```
500 PRINT :PRINT "ENTER DATA      ZZZ T
O QUIT"
510 PRINT NW;":";:INPUT T$
520 IF T$="ZZZ" THEN 590
530 A$(PW)=T$:W(NW)=PW:PW=LEN(A$):X(NW
)=PW:PW=PW+1
540 IF H=1 THEN 580
550 PRINT "HINT";:INPUT H$:IF H$="" TH
EN H=1:GOTO 580
560 GOTO 600:REM CHECK SPELLING
570 A$(PW)=H$:PW=LEN(A$):Y(NW)=PW:PW=P
W+1
580 NW=NW+1:GOTO 510
590 ND=NW-1:RETURN
600 PP=1:LT=LEN(T$)-1:LH=LEN(H$)
```

```
610 PQ=PP+LT:IF H$(PP,PQ)=T$ THEN 660
620 PP=PP+1:IF PP<=LH-LT THEN 610
630 PRINT "PLEASE CHECK SPELLING":GOTO
 550
660 IF PP=LH-LT THEN Q$="":GOTO 680
670 Q$=H$(PQ+1,LH)
680 FOR P=PP TO PQ:H$(P)="_":NEXT P
690 H$(P)=Q$:GOTO 570
900 DATA RIGHT!!,CORRECT.,YOU GOT IT!
1000 I=PEEK(764):IF I=255 THEN 1000
1010 GET #1,K:POKE 764,255
1090 I$=CHR$(K):RETURN
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

Test the "Flash" mode, and make sure it's working right. When building a complicated program, you want to make sure that everything works perfectly before adding more complexity.

Adding the Clue mode requires only a few more lines:

```
150 HN=ND:GOSUB 1300:PRINT C$;"   ";A$(
X(R)+1,Y(R))
160 PRINT :PRINT M$;:INPUT Q$:GOTO 80
```

This module hitchhikes on the "Right!!" message system set up for the "Flash" mode.

Who is in charge here?

When testing one of the advanced modes, it tries my patience to play through earlier modes. So I write temporary detours into the program to proceed directly to my desired module. For example, you go straight to Hang-Man by inserting this line

```
55 RI=99:GOTO 200
```

You can take the line out for a full-playing game at any time.

The Hang-Man module is a bit more demanding:

```
14Ø IF H=1 THEN 2ØØ
2ØØ WG=Ø:RG=Ø:HN=ND:GOSUB 13ØØ:PRINT C
$;M$;
21Ø T$=A$(W(R),X(R)):LT=LEN(T$):FOR Z=
1 TO LT
22Ø PRINT "_ ";
23Ø NEXT Z:PRINT :PRINT :PRINT M$;"GUE
SS? ";
24Ø GOSUB 1ØØØ:Z=1:PRINT :PRINT "";M$;
25Ø IF I$=T$(Z,Z) THEN 31Ø
26Ø PRINT "";
27Ø Z=Z+1:IF Z<=LT THEN 25Ø
28Ø PRINT "":GOSUB 4ØØ:IF RF=1 THEN RF
=Ø:GOTO 24Ø
29Ø WG=WG+1:IF WG<2*LT THEN 24Ø
3ØØ PRINT :PRINT "YOU LOSE":GOTO 13Ø
31Ø PRINT I$;" ";
35Ø RG=RG+1:RF=1
34Ø IF RG=LT THEN PRINT :PRINT :GOTO 1
ØØ
35Ø GOTO 27Ø
```

Lines 210 and 220 place the right number of blanks on the screen. Your guess (in line 230) gets checked against all the letters in line 260, and if it isn't found, your guess is erased (280), a strike is noted against you, and if you've missed too many (290), you lose. If, however, your guess fits into the word, the code starting at line 310 moves the cursor to the right place(s) in the display line, prints your guess, and returns for another.

It's easy to put the data inside the program or save it on a disk or tape file. The program is less flexible, but in many ways it is easier to use. The program on p. 57, TIMELINE, deals with data in the first way; after reading it, you could change SPELL ever so slightly to work the same way. And SORT (p. 86) would allow you to store several lists and call them individually into SPELL.

MATHFAX
practice math facts

A DRILLING GAME FOR FACTS

"Two and two are four; four and four are eight. . . ." The mathematical facts of life are never easy, and everyone has a pet bugaboo (for me, it's eight times seven). This program helps you perfect your grasp of those facts through drill and practice.

When we finally "know our math facts," it is usually the product of several learning systems working together: rote memory, pattern recognition, and familiarity from practice. This program helps people learn (and can be found in innumerable versions on every computer in the world) because it combines all three avenues to help us learn.

For the programmer, this is an interesting program because it offers a chance to do something often done, but with our own style. Some programmers strive to write short code—could this program be written in one line? in five? Other programmers work to make their programs "friendly" by error-trapping and hand-holding. I have spent many lines on tailoring each game to the player's abilities and offering graphic help when the answer is wrong.

MATHFAX quizzes you on your pluses, minuses, timeses, and divided-byes. It even offers a little help if you get it wrong.

```
*MATH FACTS*


 1=ADD 2=SUBTRACT 3=SUBTRACT 4=DIVIDE ? 4
 HIGHEST NUMBER FOR ADDING? 10 FOR MULTIPLY-
 ING? 7
```

You start out by tailoring the drill to your abilities: some players know more facts than others. You choose the highest operation—if you choose 3 for multiply then you get adding and subtracting too—and the highest number for adding. If you choose multiplying or dividing, you supply the highest number there, too. Some children will have mastered all their pluses to 7, for example, and will know some of the simple timeses, too—to 3, possibly. You can tailor the quiz to challenge a learner at almost any level. Adults might wish to improve their speed by setting the high numbers to 20 for addition and 13 for multiplication—that would give most everyone a workout.

```
6 * 4 = ? 24
   YES


28 / 4 = ? 8
NO

* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
```

Once you've defined the drill, you are given random problems to solve. If you answer right, the computer responds "YES" but if you are wrong, the computer draws a picture that should help you figure out the right answer for next time.

WRITING THE PROGRAM

As usual, we can sneak up on this program, and get it working a section at a time. Since the program deals with four operations, there are four different problem-makers. The main program and the addition- and subtraction-maker go like this:

```
1 REM *********************** MATHFAX *
2 DIM M$(20),Q$(1)
5 M$=CHR$(125):M$(2)="            "
10 PRINT M$;"* MATHFAX *"
```

```
20 PRINT "1=add 2=subtract 3=multiply
4=divide"
25 PRINT "highest operation";:INPUT HO
:HO=HO-1
30 PRINT "highest number for addition"
;:INPUT HA
35 IF HO<2 THEN 50
40 PRINT "             for multiplication"
;:INPUT HM
50 HN=HO:GOSUB 1300:OP=R+1:PRINT M$;:O
N OP GOTO 60,60,90,90
60 HN=HA:GOSUB 1300:N1=R:GOSUB 1300:N2
=R:ON OP GOTO 70,80
70 PRINT N1;" + ";N2;:A=N1+N2:GOTO 130
80 PRINT N1+N2;" - ";N1;:A=N2:GOTO 130
```

You need something to deal with answers:

```
130 PRINT " = ";:INPUT Q:IF Q<>A THEN
160
140 PRINT :PRINT ,,"YES"
150 FOR T=0 TO 999:NEXT T:GOTO 50
```

You need a random number maker—it is exactly like the one now appearing in programs from cover to cover of this book:

```
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

Test the program, but be sure not to ask for anything harder than addition yet. And don't make any mistakes! The program isn't ready.

```
90 HN=HM:GOSUB 1300:N1=R:GOSUB 1300:N2
=R:ON OP GOTO 70,80,100,110
100 PRINT N1;" * ";N2;:A=N1*N2:GOTO 13
0
110 IF N1=0 OR N2=0 THEN 90
120 PRINT N1*N2;" / ";N1;:A=N2:GOTO 13
0
```

Dividing by zero is risky business; in fact, zero is tricky anywhere in division, so line 110 rejects any problems with zeroes in them.

Test again—using all the operations—but you still can't make any mistakes. If you do, the programs crashes. (Oh, go on, make a mistake, just to see what happens.)

The capabilities of the ATARI are hardly touched by this little program, so let's add a little bit of help. Most people don't like tests unless right answers and maybe even hints are supplied when you miss.

```
160 PRINT :PRINT ,,"NO":ON OP GOTO 170
,165,180,180
165 NT=N1+N2:GOTO 175
170 NT=N2
175 PRINT ,;:GOSUB 300:PRINT ,;:NT=N1:
GOSUB 300:GOTO 150
180 NZ=0
190 IF NZ<N1 THEN PRINT ,:NT=N2:GOSUB
300:NZ=NZ+1:GOTO 190
250 PRINT ,,"Okay";:INPUT Q$:GOTO 50
300 NP=0
310 IF NP<NT THEN PRINT CHR$(0);:NP=NP
+1:GOTO 310
320 PRINT :RETURN
```

For addition and subtraction, two lines of beads are drawn on the screen. To get the right answer in addition, you count all the beads. In subtraction, you only count the beads that aren't paired in two lines of beads. For multiplication, a rectangle of beads is drawn. To get the right answer in multiplication you count all the beads—you may know a shortcut, called a "times-table." In division, you know the number of beads in one side, and the total number of beads, so you just need to count the side you don't know.

ADVANCED STUDIES

A timer and a scorekeeper would make this program more challenging for those of us who feel confident of our math facts. It would be an easy matter to borrow them from REMEMBER (p. 43). You would need to change the "keyboard" at line 130 to one that uses GET I$ and includes a timing loop.

TIMELINE
historical events

A TIME MACHINE

The "mind of the computer" is sometimes compared to the human mind. The comparison is partly right. Microcomputers have tens of thousands of memory locations—the ATARI has more than 48,000, lined up in one long row, like links in a chain. Each link can hold (and always does) one of 256 symbols. History is the story that ties a long row of events together. The microcomputer's memory can easily be made into a perfect model of events in history. This program roves over the terrain of dates and events like a time machine we get to drive!

Writing the program—and understanding the concepts behind the program—are the best part of this chapter. The part of the human mind lacking in the computer is the "relationship finder" we humans use to spot linkages between two apparently unrelated facts. The computer provides a smooth magic carpet to survey the facts, but it is not very good at noting interesting coincidences. My computer never said, "Did you know that the first steam-powered boat didn't run up the Hudson River until 32 years after Watt invented the steam-engine? Is it my imagination or are things speeding up?"

For a machine, whether made of flesh and blood or silicon and gold, to make observations like that takes a lot of smarts. Possibly someone who builds TIMELINE's simple magic historical carpet may later make key breakthroughs in teaching intelligence to machines.

```
                * TIMELINE *

          1000
   Leif Ericson discovers America
          1522
   first circumnavigation of the earth
          1607
   Jamestown settled in Virginia
          1620
   Pilgrims arrive at Plymouth Rock
          1756
   French and Indian War begins
     -  -  -  -  -  -  -  -  -  -  -
   - earlier    + later     What next?
```

TIMELINE provides a skeleton for a "terrain" of historical events for you to explore. You can see what happened last, happens next, what happened 10 years ago, what happens in 20 years—assuming you programmed the computer to know the answer. Now that I think of it, the computer wouldn't mind if you made up a history of the future.

WRITING THE PROGRAM

The program reads up to 100 dates and events and stores them in memory (lines 10 and 20). The main program begins at line 40, where it displays five adjacent events:

```
1 REM ********************* TIMELINE *
2 DIM C$(24),T$(39),A$(4000),K$(1)
3 DIM Y(100),Z(100)
4 OPEN #1,4,0,"K:"
5 C$=CHR$(125):C$(2)="         * TIMEL
INE *":PRINT C$
9 N=0:P=0
10 Z(N)=P:READ YR,T$:IF T$="ZZZ" THEN
30
20 LT=LEN(T$):A$(P+1)=T$:N=N+1:Y(N)=YR
:LT=LEN(T$):P=P+LT:GOTO 10
30 LN=N:FOR N=1 TO LN:PRINT Y(N):PRINT
 A$(Z(N-1)+1,Z(N)):PRINT :NEXT N:Y=3
```

```
40 PRINT C$:FOR N=Y-2 TO Y+2:PRINT Y(N
):PRINT A$(Z(N-1)+1,Z(N)):PRINT :NEXT
N
50 PRINT " -  -  -  -  -  -  -  -  -
-  -  -"
60 PRINT "   - earlier  + later  which
?";:GOSUB 1000:Y=Y+1
70 IF K$="+" THEN Y=Y+3
75 IF Y>LN-2 THEN Y=LN-2
80 IF K$="-" THEN Y=Y-5
85 IF Y<3 THEN Y=3
90 GOTO 40

1000 I=PEEK(764):IF I=255 THEN 1000
1010 GET #1,K:POKE 764,255
1020 K$=CHR$(K)
1090 RETURN
```

Lines 60-70 find out what you want to do next (using a keyboard subroutine at line 1000). Lines 80 and 90 make sure you don't run out of facts, then loop you back to a new display in line 40.

There is lots of room for data in lines 100-999. Be sure you put your favorite facts in order.

If you don't have any favorite facts, here are a few of mine to help show you the format:

```
100 DATA 1000,Leif Ericson discovers A
merica
110 DATA 1522,first circumnavigation o
f the earth
120 DATA 1607,Jamestown settled in Vir
ginia
130 DATA 1620,Pilgrims arrive at Plymo
uth Rock
140 DATA 1756,French & Indian War begi
ns
150 DATA 1775,James Watt invents steam
 engine
```

```
160 DATA 1776,Declaration of Independe
nce
170 DATA 1778,Captain Cook discovers H
awaii
180 DATA 1787,U.S. Constitution signed
190 DATA 1791,Bill of Rights ratified
200 DATA 1803,Louisiana Purchase
210 DATA 1807,Robert Fulton's steamboa
t up the Hudson
220 DATA 1834,Charles Babbage's analyt
ical engine
230 DATA 1846,Potato famine & U.S.-Mex
ico war
240 DATA 1849,California Gold Rush
250 DATA 1858,first trans-Atlantic cab
le
260 DATA 1861,Civil War begins
270 DATA 1865,Abraham Lincoln shot
280 DATA 1869,Golden spike: railroad a
cross U.S.
290 DATA 1888,George Eastman invents K
odak camera

999 DATA -1,ZZZ
```

The last data line must end with "ZZZ," or else the computer will give you an "out of data" error.

That completes the program. Test, debug, and save it.

Of course, it would be a simple matter to use a program like SSORT to enter data, sort it, and save it, then modify this program so it could be read from a storage device.

MINI-WOMBATS
for word problems

I want computer games to surprise and delight me. I invented Wombats* watching children "tricking" computers by typing in nonsense words and false names. Peals of laughter greeted

```
WELL DONE, BRTLSPYX!

NOW TRY 3 + 5 = ??
```

Mini-WOMBATS is a small—but expandable!—word-problem generating game intended for experienced and/or patient programmers. WOMBATS is a small expert system; its expertise lies in building simple English stories that ask number questions. Its stories include the player, a friend, and an object of her choice. Which points out a problem a computer might have telling stories: gender! It is OK for the player to make up joke names and objects, but the computer should speak as elegantly as it can. Pronouns and verbs should agree with their nouns. (In this program, some do, and some don't).

The string-handling in this program is intense, hinting at the incredible complexity that natural language programs will need to understand the languages of humans.

The player chooses the names of the people and the things in the stories, and then the computer makes up number problems about them.

At the beginning of play, the highest level of operations (1 = add to 4 = divide), the highest number for addition, and, if needed, the highest number for multiplication, are chosen.

*The original version appeared in *Creative Computing* in October, 1981, page 216.

Sample dialogue (player dialogue in italics):

```
Please tell me your name? Damiana

= + = + = + = + = + = + =
Sienna found a bag containing 7 wombats. She already has
5 wombats at home. How many does she have now? 12

You got it!

——— ——— ——— ——— ——— ——— ——— ——— ———
Name a person? bob
Person's names begin with a CAPITAL.
Remember the SHIFT key, and
please try again, Damiana.

——— ——— ——— ——— ——— ——— ——— ——— ———
Name a person? Bob
Is Bob a boy or a girl? boy

++ ++ ++ ++ ++ ++ ++ ++ ++
Name an object? flat bed truck

= + = + = + = + = + = + =
Bob had 13 flat bed trucks in his pocket, and later won 15
more in a bet with Damiana. How many did he have then?
28

Well done.

= + = + = + = + = + = + =
Bob dug 4 flat bed truck traps in the forest, and caught a
total of 36 flat bed trucks. On the average, how many did he
find in each trap? 9
```

WRITING THE PROGRAM

Write this program in several sittings, carefully saving its parts after each session. The program is long and takes precise typing—kind of like putting a computer together from a kit.

The framework of the program begins with the introduction, which prepares the computer for the problems and tailors the problems to the player:

```
1 REM ****************** miniWOMBATS *
2 DIM Q$(25Ø),N$(16),J$(32),P$(32),SX$
(12),C$(1)
5 C$=CHR$(125):PRINT C$
8 POKE 7Ø2,Ø
9 SX=1:P$="Sienna":J$="wombat"
1Ø PRINT :PRINT "* Wombats !! *":LN=Ø
2Ø PRINT "1=add 2=subtract 3=multiply
4=divide"
25 PRINT "highest operation";:INPUT HO
:HO=HO-1
3Ø PRINT "highest number for addition"
;:INPUT HA
35 IF HO<2 THEN 5Ø
4Ø PRINT "            for multiplication"
;:INPUT HM
5Ø PRINT C$:PRINT :PRINT :PRINT
53 PRINT "Please tell me your name";:I
NPUT N$
55 IF ASC(N$)>9Ø THEN GOSUB 117Ø:GOTO
53
6Ø HN=3:GOSUB 13ØØ:IF R=3 THEN GOSUB 1
1ØØ
7Ø GOSUB 13ØØ:IF R=3 THEN GOSUB 12ØØ
8Ø HN=HO:GOSUB 13ØØ:OP=R+1:IF OP>2 THE
N HN=HM:GOTO 9Ø
85 HN=HA
9Ø GOSUB 13ØØ:N1=R:GOSUB 13ØØ:N2=R:ON
OP GOTO 1ØØ,3ØØ,5ØØ,7ØØ
```

This program requires lower-case, so the POKE in line 8 makes sure they are on.

The addition problems are next. Lots of string handling. The program builds strings using this technique:

```
A$: Sienna found a bag containing
                            LEN(A$)
```

You can add more to A$ right here and that position is LEN(A$)+1

```
100 HN=2:GOSUB 1300:ON R+1 GOTO 110,17
0,220
110 Q$=P$:GOSUB 1500:Q$(Z)=" found a b
ag containing ":GOSUB 1500
120 Q$(Z)=STR$(N1):GOSUB 1500:Q$(Z)="
":Q$(Z+1)=J$:GOSUB 1500
130 Q$(Z)="s. ":GOSUB 1400
140 GOSUB 1500:Q$(Z)="already has ":GO
SUB 1500:Q$(Z)=STR$(N2):GOSUB 1500
150 Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500:Q$
(Z)="s at home. How many does "
160 GOSUB 1420:GOSUB 1500:Q$(Z)="have
now":GOTO 290
170 Q$=P$:GOSUB 1500:Q$(Z)=" had ":GOS
UB 1500:Q$(Z)=STR$(N1):GOSUB 1500
180 Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500:Q$
(Z)="s in ":GOSUB 1480:GOSUB 1500
190 Q$(Z)="pocket, and later won ":GOS
UB 1500:Q$(Z)=STR$(N2):GOSUB 1500
200 Q$(Z)=" more in a bet with ":GOSUB
 1500:Q$(Z)=N$:GOSUB 1500
210 Q$(Z)=". How many did ":GOSUB 1420
:GOSUB 1500:Q$(Z)="have then":GOTO 290
220 Q$=P$:GOSUB 1500:Q$(Z)=" receives
2 envelopes. One contains ":GOSUB 1500
230 Q$(Z)=STR$(N1):GOSUB 1500:Q$(Z)="
":Q$(Z+1)=J$:GOSUB 1500
240 Q$(Z)="s; the other contains ":GOS
UB 1500:Q$(Z)=STR$(N2):GOSUB 1500
250 Q$(Z)=". How many does ":GOSUB 142
0:GOSUB 1500:Q$(Z)="have now":GOTO 290
290 A=N1+N2:GOTO 900
```

P$ is the name of the current person, J$ is the name of the object. STR$(n1) changes a number into a text string—you may not see any difference, but the computer does.

The lines from 900 to 1090 deal with right and wrong answers:

```
900 PRINT :PRINT " = + = + = + = + = +
 = + = + = + =":PRINT
910 LQ=LEN(Q$):L1=1:L2=36
920 IF Q$(L2,L2)=" " THEN 940
930 L2=L2-1:GOTO 920
940 PRINT Q$(L1,L2):L1=L2+1:L2=L1+34:I
F L2<LQ THEN 920
950 PRINT Q$(L1,LQ);"";
960 INPUT Q:IF Q=A THEN 1000
970 PRINT :PRINT "Woops! That's not ri
ght."
980 PRINT "The right answer is ";A;"."
990 GOTO 1090
1000 HN=3:GOSUB 1300:ON R GOTO 1010,10
20,1030
1010 PRINT "You got it!":GOTO 1090
1020 PRINT "Well done, ";N$;".":GOTO 1
090
1030 PRINT "That's right!":GOTO 1090
1090 FOR LS=1 TO 12:PRINT :NEXT LS:PRI
NT "";:GOTO 60
```

The routines that get new persons and objects go like this:

```
1100 PRINT :PRINT " ++ ++ ++ ++ ++ ++
++ ++ ++ ++ ++ ++"
1110 PRINT "Name a person";:INPUT P$
1120 IF ASC(P$)>90 THEN GOSUB 1170:GOT
O 1110
1130 PRINT "Is ";P$;" a girl or a boy"
;:INPUT SX$:SX=0:IF SX$="boy" THEN SX=
2
1140 IF SX$="girl" THEN SX=1
1150 IF SX>0 THEN RETURN
1160 PRINT "I'm sorry. I don't know an
y questions":PRINT "about ";SX$;"s. Pl
ease try again.":GOTO 1130
```

```
1170 PRINT "Person's names begin with
a CAPITAL."
1180 PRINT "Remember the SHIFT key, an
d please"
1190 PRINT "try again, ";N$;".":RETURN

1200 PRINT :PRINT " -- -- -- -- -- --
-- -- -- -- -- --"
1210 PRINT "Name an object";:INPUT J$
1290 RETURN
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

You spotted line 1300—a familiar face in the crowd. Subroutines in the 1400s supply necessary pronouns, like this:

```
1400 GOSUB 1500:IF SX=1 THEN Q$(Z)="Sh
e ":RETURN
1410 Q$(Z)="He ":RETURN
1420 GOSUB 1500:IF SX=1 THEN Q$(Z)="sh
e ":RETURN
1430 Q$(Z)="he ":RETURN
1440 GOSUB 1500:IF SX=1 THEN Q$(Z)="he
rs ":RETURN
1450 Q$(Z)="his ":RETURN
1460 GOSUB 1500:IF SX=1 THEN Q$(Z)="he
r ":RETURN
1470 Q$(Z)="him ":RETURN
1480 GOSUB 1500:IF SX=1 THEN Q$(Z)="he
r ":RETURN
1490 Q$(Z)="his ":RETURN
1500 Z=LEN(Q$)+1:RETURN
```

With these modules intact, you should be able to test the program, but only in first gear: addition. When you have everything operating correctly, we can move on to the three other operations.

Subtraction:

```
300 N2=N1+N2:HN=2:GOSUB 1300:ON R+1 GO
TO 310,340,400
310 Q$=P$:GOSUB 1500:Q$(Z)=" hides ":G
OSUB 1500:Q$(Z)=STR$(N2):GOSUB 1500
320 Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500:Q$
(Z)="s, and you find ":GOSUB 1500
330 Q$(Z)=STR$(N1):GOSUB 1500:Q$(Z)=".
 How many are still hidden":GOTO 490
340 Q$=P$:GOSUB 1500:Q$(Z)=" had too m
any ":GOSUB 1500:Q$(Z)=J$:GOSUB 1500
350 Q$(Z)="s and gave you ":GOSUB 1500
:Q$(Z)=STR$(N2):GOSUB 1500
360 Q$(Z)=". Later, ":GOSUB 1420:GOSUB
 1500:Q$(Z)="lost all of them, "
370 GOSUB 1500:Q$(Z)="and you kindly g
ave ":GOSUB 1500
380 Q$(Z)=STR$(N1):GOSUB 1500:Q$(Z)="
back. How many do you still have":GOTO
 490
400 Q$="Yesterday, ":GOSUB 1500:Q$(Z)=
P$:GOSUB 1500:Q$(Z)=" bought ":GOSUB 1
500
410 Q$(Z)=STR$(N2):GOSUB 1500:Q$(Z)="
":Q$(Z+1)=J$:GOSUB 1500
420 Q$(Z)="s, but this morning ":GOSUB
 1420:GOSUB 1500
430 Q$(Z)="could only find ":GOSUB 150
0:Q$(Z)=STR$(N1):GOSUB 1500
440 Q$(Z)=". How many were missing":GO
TO 490
490 A=N2-N1:GOTO 900
```

Multiplication:

```
500 HN=2:GOSUB 1300:ON R+1 GOTO 510,56
0,600
510 Q$=P$:GOSUB 1500:Q$(Z)=" won ":GOS
UB 1500:Q$(Z)=STR$(N1):GOSUB 1500
520 Q$(Z)=" coupons at the fair. ":GOS
UB 1400:GOSUB 1500
530 Q$(Z)="exchanged each coupon for "
:GOSUB 1500:Q$(Z)=STR$(N2):GOSUB 1500
540 Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500:Q$
(Z)="s. How many does ":GOSUB 1420
550 GOSUB 1500:Q$(Z)="have now":GOTO 6
90
560 Q$=P$:GOSUB 1500:Q$(Z)=" built a m
achine to make ":GOSUB 1500
570 Q$(Z)=J$:GOSUB 1500:Q$(Z)="s. It h
as made ":GOSUB 1500
580 Q$(Z)=STR$(N1):GOSUB 1500:Q$(Z)="
each day for ":GOSUB 1500:Q$(Z)=STR$(N
2)
590 GOSUB 1500:Q$(Z)=" days. How many
has it made":GOTO 690
600 Q$="A flying saucer deposits ":GOS
UB 1500:Q$(Z)=STR$(N1):GOSUB 1500
610 Q$(Z)=" silvery spheres in ":GOSUB
 1500:Q$(Z)=P$:GOSUB 1500
620 Q$(Z)="'s back yard, and ":GOSUB 1
500:Q$(Z)=STR$(N2):GOSUB 1500

630 Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500:Q$
(Z)="s jumped out of each one. How man
y "
640 GOSUB 1500:Q$(Z)=J$:GOSUB 1500:Q$(
Z)="s are rampaging around ":GOSUB 150
0
650 Q$(Z)=P$:GOSUB 1500:Q$(Z)="'s hous
e right now":GOTO 690
690 A=N1*N2:GOTO 900
```

Division:

```
700 IF N1=0 THEN HN=HM:GOSUB 1300:N1=R
:GOTO 700
710 N2=N1*N2:HN=2:GOSUB 1300:ON R+1 GO
TO 720,770,820
720 Q$=P$:GOSUB 1500:Q$(Z)=" dug ":GOS
UB 1500:Q$(Z)=STR$(N1):GOSUB 1500
730 Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500:Q$
(Z)=" traps in the forest, and caught
a total of "
740 GOSUB 1500:Q$(Z)=STR$(N2):GOSUB 15
00:Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500
750 Q$(Z)="s. On the average, how many
 did ":GOSUB 1420:GOSUB 1500
760 Q$(Z)="find in each trap":GOTO 890
770 Q$=P$:GOSUB 1500:Q$(Z)=" buys a pa
cket with ":GOSUB 1500:Q$(Z)=STR$(N2)
780 GOSUB 1500:Q$(Z)=" ":Q$(Z+1)=J$:GO
SUB 1500:Q$(Z)=" seeds in it. The dire
ctions tell "
790 GOSUB 1460:GOSUB 1500:Q$(Z)="to pl
ant ":GOSUB 1500:Q$(Z)=STR$(N1):GOSUB
1500
800 Q$(Z)=" in each hole. How many hol
es should ":GOSUB 1420:GOSUB 1500
810 Q$(Z)="dig":GOTO 890
820 Q$=P$:GOSUB 1500:Q$(Z)=" shared ":
GOSUB 1500:Q$(Z)=STR$(N2):GOSUB 1500
830 Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500:Q$
(Z)="s with ":GOSUB 1500:Q$(Z)=STR$(N1
-1)
840 GOSUB 1500:Q$(Z)=" of ":GOSUB 1480
:GOSUB 1500
850 Q$(Z)="friends. How many did each
get":GOTO 890
890 A=N2/N1
```

I left one particularly glaring grammatical error in: Bob had 1 flat bed trucks in his pocket. . . ." Could someone please write a few lines of code to trap that error?

There is room for adding more questions, and a simple score-keeping section, with a display of the score from time to time. The original version of the program also had a diagnostic section for teachers: the ability to print the problems as they were solved and summaries of problems solved, and an extensive HELP capability. Interested programmers should look up that classic October 1981 edition of *Creative Computing.*

the computer as a tool

The last six chapters are about tools you can build to use your computer for more serious (and helpful) jobs: as a number and data-cruncher.

The programming is no trickier than in earlier chapters, but some of the programs often consist of more modules, complex looping, and other precise code. If at first it makes no sense, read the chapter over again. Sleep on it. Understanding will dawn.

Be prepared to start thinking about all the great ways you can rewrite these programs to do all manner of other useful things. The nicest thing about computers is that they are willing to try anything twice.

PAINT
electronic crayons

The computer makes a wonderful set of electronic color crayons. And it's fun to see our own pictures on the TV screen for a change! Using this program, anybody can doodle.

For the programmer, there are two challenges: this program fiddles with color commands and saves and retrieves data with tape or disk. Fortunately, you can take on one challenge at a time.

They say a picture is worth a thousand words.

PAINT gives the player control over the color video capabilities of the ATARI, to create pictures and illustrations using colored squares. This turns out to be an enjoyable way to gain mastery over the ATARI's cursor control keys, while making pretty pictures.

PLAYING THE GAME

When the program starts, your "brush" is the block in the middle of the "paper." You can move it up, down, left, and right, using the cursor keys in the lower right-hand corner of the keyboard. To get the "brush" where you want it takes practice.

Fortunately, you can practice moving the brush all you want without making a mark on the paper. When you're ready for paint, you press the space bar, and the brush begins depositing color wherever it moves. You can change the brush's color by pressing the number keys at the top of the keyboard—only 1, 2, 3, and 0 work, because there are only four colors in this GRAPHICS mode. Color 0 is the background color, and is useful for erasing.

After some practice, you will be able to interlace colors easily, turning your brush on and off using the space bar with precision. If you've spoiled a painting beyond retrieval, you can zap it—just press Z and then Y.

You can save a painting by pressing <S>. There is lengthy number-crunching involved, turning the colored picture into terms the computer knows how to save—it takes about a minute, so be patient. You can tell where the crunching is taking place by the line of blocks descending the left margin of your picture. As soon as the crunching is done, the computer asks for a name for your picture.

ATARI notes

Saving files to disk

When choosing a name for a painting, program, or anything else you plan to save, it's a good idea to make the name as descriptive as possible, so when you look at the disk directory you will know what each file is. And another thing: when you plan to save several versions of a file, number them—BOAT1, BOAT2, BOAT3—and keep track of the last number you used.

Saving files to tape

The programs in this book use the input/output commands for a disk drive system, but all can be adapted to work with tape storage, too. Tape commands are usually simpler than disk commands. Please check the details in your ATARI manuals.

If you have a painting saved, you can load it by pressing <L>. The program asks for the name of the file to load. (If you tell it a file that isn't there, it will give you an error message.)

You can adjust the color and luminance of your four colors by pressing <C>. The current color will be changed to grey. Press the <+> and <-> keys until you have found your desired color, then press <RETURN> to accept it. The color will then be changed to the lowest luminance. Again, press <+> and <-> until you have the desired brightness, then accept it by pressing <RETURN>.

You can quit the program at any time by pressing <Q>.

WRITING THE PROGRAM

The main framework of PAINT is quite simple.

```
1 REM ************************* PAINT *
2 GM=3:MX=39:MY=23
3 DIM K$(1),E$(8),F$(14),S$(MX),P$(MX*
(MY+1))
4 OPEN #1,4,Ø,"K:"
5 GRAPHICS GM+16:X=MX/2:Y=MY/2:PC=1:CO
LOR 1
9 CF=1
1Ø CF=PC:GOSUB 11ØØ:GOSUB 1ØØØ
2Ø IF K=45 THEN Y=Y-1:IF Y<Ø THEN Y=Ø
3Ø IF K=61 THEN Y=Y+1:IF Y>MY THEN Y=M
Y
4Ø IF K=43 THEN X=X-1:IF X<1 THEN X=1
5Ø IF K=42 THEN X=X+1:IF X>MX THEN X=M
X
6Ø IF K=32 AND PF=1 THEN PF=Ø:GOTO 7Ø
65 IF K=32 THEN PF=1
7Ø IF K>47 AND K<52 THEN PC=K-48:CF=PC
 19Ø GOTO 1Ø
```

The first few lines clear the screen and set the brush in the middle of the paper. The main program starts at line 10. The group of lines which all begin IF K= deal with each possible key pressed—lines 20-50—decode the cursor keys, and change the X-Y position of the brush in accordance with the player's commands. Lines 60 and 65 toggle—switch on and off—the brush. Line 70 takes a number and translates it into a new paint color.

```
1ØØØ I=PEEK(764):IF I=255 THEN 1ØØØ
1Ø1Ø GET #1,K:POKE 764,255
1Ø3Ø IF PF=Ø THEN CF=OC
1Ø4Ø IF PF=1 THEN CF=PC
1Ø5Ø GOSUB 11ØØ
1Ø8Ø K$=CHR$(K)
1Ø9Ø RETURN
11ØØ LOCATE X,Y,OC:COLOR CF:PLOT X,Y:D
RAWTO X,Y
119Ø RETURN
```

Subroutine 1000 is the "keyboard," which gets the next command from the player. Subroutine 1100 places a brushstroke on the screen.

With this much of the program entered, you can make pictures, and make sure everything is working right so far. Remember: in programming, a keystroke out of place is like a needle in a haystack. The real fun is getting clues from a program that is not quite working and eradicating those stubborn bugs.

The save module contains some fancy business to accommodate the ATARI's storage scheme: the painting must be translated into a long string of numbers that can be stored like text.

```
300 GOSUB 400
310 PRINT "FILE TO SAVE";:INPUT E$
320 F$="D:":F$(3)=E$:F$(LEN(F$)+1)=".P
IX"
330 OPEN #2,8,0,F$
340 FOR SY=0 TO MY:PP=1+MX*SY:S$=P$(PP
,PP+38):PRINT #2;S$:NEXT SY
350 PRINT #2;"ZZZ":CLOSE #2
390 LX=MX:LY=MY:GOSUB 600:GOTO 10
400 PP=1:FOR SY=0 TO MY:FOR SX=1 TO MX
410 LOCATE SX,SY,P:P$(PP)=STR$(P):PP=P
P+1
420 NEXT SX
430 PLOT 1,SY:COLOR 1:DRAWTO 1,SY
440 NEXT SY
490 RETURN
```

If you can save, you should be able to load, too:

```
500 PRINT "FILE TO LOAD";:INPUT E$
510 F$="D:":F$(3)=E$:F$(LEN(F$)+1)=".P
IX"
520 OPEN #2,4,0,F$:LY=0:PP=1
540 INPUT #2;S$:IF S$="ZZZ" THEN 580
550 P$(PP)=S$:LX=LEN(S$):PP=PP+LX
560 LY=LY+1:GOTO 540
580 CLOSE #2:LY=LY-1
590 GOSUB 600:GOTO 10
600 GRAPHICS GM+16:PP=1:FOR SY=0 TO LY
:FOR SX=1 TO LX
```

```
610 CF=VAL(P$(PP,PP)):COLOR CF
620 PLOT SX,SY:DRAWTO SX,SY:PP=PP+1
630 NEXT SX:NEXT SY
690 RETURN
```

We need to be able to clear the screen for a new painting, so a zapper looks like this:

```
800 GOSUB 400:PRINT "ZAP IT";
810 INPUT K$:IF K$="N" THEN LX=MX:LY=M
Y:GOSUB 600:GOTO 10
890 GRAPHICS GM+16:GOTO 10
```

You also need to be able to quit:

```
900 GOSUB 400:PRINT "QUIT":INPUT K$:IF
 K$="N" THEN LX=MX:LY=MY:GOSUB 600:GOT
O 10
990 END
```

And changing color and luminance is accomplished by these lines:

```
200 CV=0:PV=PC-1:IF PV<0 THEN PV=4
210 SETCOLOR PV,CV,6:GOSUB 1000:IF K=1
55 THEN 250
220 IF K$="+" THEN CV=CV+1:IF CV>15 TH
EN CV=15
230 IF K$="-" THEN CV=CV-1:IF CV<0 THE
N CV=0
240 GOTO 210
250 LV=0
260 SETCOLOR PV,CV,LV:GOSUB 1000:IF K=
155 THEN 10
270 IF K$="+" THEN LV=LV+2:IF LV>14 TH
EN LV=14
280 IF K$="-" THEN LV=LV-2:IF LV<0 THE
N LV=0
290 GOTO 260
 80 IF K$="Q" THEN 900
 90 IF K$="Z" THEN 800
```

```
100 IF K$="S" THEN 300
110 IF K$="L" THEN 500
120 IF K$="C" THEN 200
```

ADVANCED TOPICS

Another way to "save" a picture is by photographing it. You need to have a camera that can be set for long exposures, because the light from a TV is weak, and a short exposure may have bars across the screen.

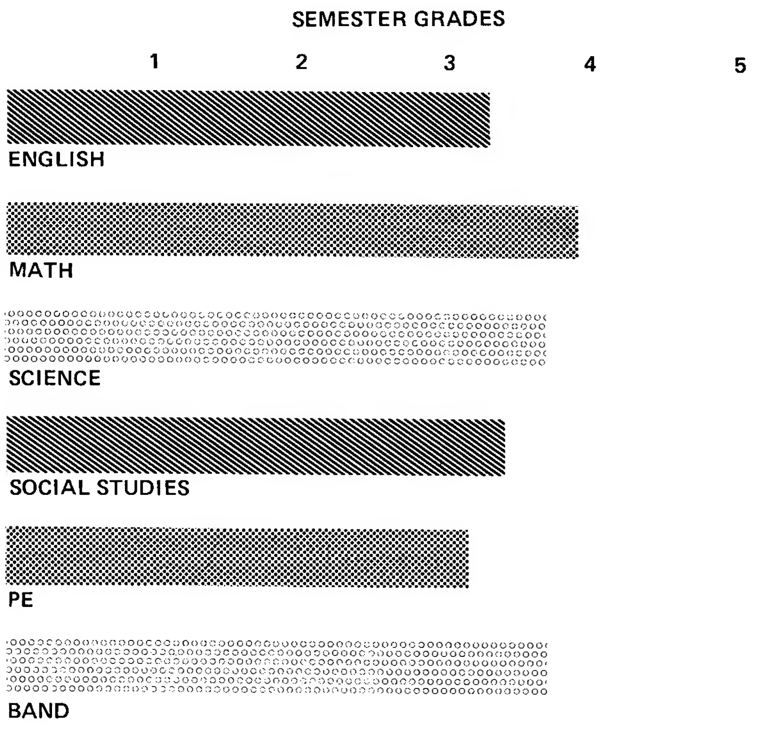
Before you push the shutter-release button, make sure of the following points:

- Eliminate screen glare. (If you can see a reflection of a window, or a wall, or yourself, you can be sure it will be in the photograph, too. Change the lighting. Taking your picture after dark in a darkened room may be the best way.)
- Fill the frame and make sure it's sharply focused. (If you can't get close enough, don't waste film. Borrow a camera with a longer lens.)
- Use a tripod or make a steady stand for your camera. (Humans move a lot in half a second. You may even need to ask people to be motionless while you make your picture, so the house doesn't shake.)
- Adjust your TV for its crispiest, juiciest image. (What looks good through the camera may be different than what's easiest to work with on the screen.)
- Bracket your exposures. Try several different exposure times, from as short as 1/60th of a second to as long as 2 seconds. Experience is the best teacher.

BARS

plot data with a bar graph

Computer graphics are exciting, as we can tell from prime-time TV, fantasy movies, and arcade games. Plotting graphs is a less exciting cousin of the flashy displays we sometimes see, but it is an excellent use for the computer. It is often easier to see relationships and judge their meaning if we can view them graphically:



Get the picture? For the programmer, this program is a breeze. It includes a new trick—using the printer—but it works fine without a printer.

BARS lets the user translate numerical data into bar graphs on the computer screen. The screen can handle up to seven bars, and a large range of possible numbers, and graphically represent the relationships between the different classes.

There are four possible different versions of BARS, like this:

	Input	*Data*
screen	BARS	BARS2
printer	BARS3	BARS4

The screen version displays the graph on the video screen, while the printer version prints it on a printer. The INPUT version asks the user to put the data in each time the program is run; in the DATA version the data is part of the program, so a graph is drawn automatically when the program is RUN.

All versions of the program require scaling information: what is the lowest number you will show on the graph? the highest number? How many bars will you show? and what is the overall title for your graph? The program works with data pairs—a title, and a number, like

```
USA,69
```

which represents the life expectancy of a male born in the United States. When you sit down at the computer to generate a bar graph, you will need to know all the data.

WRITING THE PROGRAM

The second version is the easiest on the programmer, because you won't have to keep giving the computer data for the graph, so we start writing there:

```
1 REM ************************** BARS *
2 DIM C$(40),B$(7),T$(38),S$(38),A$(26
6)
3 DIM L(7),F(7)
5 C$=CHR$(125):C$(2)="          * BARS *"
:PRINT C$
```

```
10 READ B:IF B=-1 THEN 30
20 B$(LEN(B$)+1)=CHR$(B):GOTO 10
30 READ LN
40 READ HN
50 READ NB
60 READ T$:LT=LEN(T$):M=(38-LT)/2:C$(1
+M)=T$
70 P=0:N=0
80 READ S$,V:L(N)=V:LS=LEN(S$):P(N)=P:
A$(P+1)=S$:P(N)=P:P=P+LS:N=N+1:IF N<NB
 THEN 80
90 P(N)=P
```

This section is a simple series of READs that get DATA from anywhere in the program. Lines 10 and 20 read the characters that get translated into bars on your video screen. Lines 30-90 read the data for your particular graph—low number, high number, number of bars, title string, and an individual title-string for each bar.

We can build a program in any order, so let's do the DATA next:

```
900 DATA 0,8,10,19,79,88,123,-1
910 DATA 0,75,6
920 DATA LIFE EXPECTANCY - MALES
930 DATA USA,69
940 DATA COLOMBIA,44
950 DATA NORWAY,71
960 DATA JAPAN,71
970 DATA INDIA,42
980 DATA AUSTRALIA,68
```

Line 900 contains the codes for the letters that make up the bars. The -1 at the end tells the READ statement in line 10 that it has read all the color controls and can go on to the next part. Line 910 contains the low number, high number, number of bars, and title. The next six lines contain the specific titles and data. (Of course, there is nothing magical about my data—use any data you want.)

The bar-drawing code is next:

```
100 POKE 752,1:PRINT C$:PRINT
110 I=(HN-LN)/5:IB=36/(HN-LN)
120 FOR N=0 TO 5:PRINT INT(LN+(N*I));"
     ";:NEXT N:PRINT
130 PRINT :FOR BN=0 TO NB-1
140 FOR L=1 TO L(BN)*IB:PRINT B$(BN+1,
BN+1);:NEXT L:PRINT " ";L(BN)
150 PRINT A$(P(BN)+1,P(BN+1)):PRINT :N
EXT BN

200 GOTO 200
```

Lines 120-130 draw the reference line that tells what the bars represent. If these numbers turn out strangely, you can adjust the high number and low number to make them more pleasing. The POKE in line 100 turns off the cursor, so all you see on the screen is your graph.

By the way, what in the world does line 200 do? It's easy to find out: take it out, RUN the program, and see what happens.

The program is done. Test and debug it before we move on to three more versions. And be sure to save this version if you want to use it again.

To make the "plain vanilla" version of BARS, all you need to do is change six lines at the beginning of the program, so they look like this:

```
30 PRINT "LOW NUMBER";:INPUT LN
40 PRINT "HIGH NUMBER";:INPUT HN
50 PRINT "NUMBER OF BARS";:INPUT NB
60 PRINT "TITLE FOR GRAPH";:INPUT T$:L
T=LEN(T$):M=(38-LT)/2:C$(1+M)=T$
70 P=0:N=0
80 PRINT "BAR ";N+1;" TITLE";:INPUT S$
:LS=LEN(S$):P(N)=P:A$(P+1)=S$:P(N)=P:P
=P+LS
85 PRINT "         VALUE";:INPUT V:L(N)=V
:N=N+1:IF N<NB THEN 80
90 P(N)=P
```

This version of the program allows you to explore the bar-graphing possibilities and plan your graphs carefully before saving them.

PRINTING THE BAR GRAPH

```
SEMESTER GRADES

Ø     1     2     3     4     5

8888888888888888888888888 3.6
ENGLISH

############################ 4
MATH

********************** 3
SOC SCI

HHHHHHHHHHHHHHHHHHHHHHHH 3.3
PE

++++++++++++++++++++++++++++ 4
CHEMISTRY

XXXXXXXXXXXXXXXXXXXXXXXXX 3.6
SPANISH
```

It is easy to "hardcopy" your graph with a printer. It just takes a print module:

```
24Ø LPRINT :FOR BN=Ø TO NB-1:T$=""
25Ø FOR L=1 TO L(BN)*IB:T$(LEN(T$)+1)=
P$(BN+1,BN+1):NEXT L:T$(LEN(T$)+1)="  "
:T$(LEN(T$)+1)=STR$(L(BN)):LPRINT T$
26Ø LPRINT A$(P(BN)+1,P(BN+1)):LPRINT
:NEXT BN
```

```
900 DATA 0,8,10,19,79,88,123,-1
990 DATA 56,35,42,72,159,88,78,-1

200 PRINT "PRINT IT";:INPUT Q$:N=0:LPR
INT T$:LPRINT
210 READ P:IF P=-1 THEN 230
220 P$(N+1)=CHR$(P):N=N+1:GOTO 210
230 T$="":FOR N=0 TO 5:T$(LEN(T$)+1)=S
TR$(INT(LN+(N*I))):T$(LEN(T$)+1)="
 ":NEXT N:LPRINT T$
```

Line 200 replaces the earlier line (that swallowed its own tail) with a query. Lines 210 and 220 read the codes to make graphic blocks on the printer. Line 990 contains those codes.

Now that you have three versions of the program, it's an easy matter to borrow the lines between 200 and 300—and don't forget line 990. While you're about it, you might play with some of the other codes your printer can make—you might think they look better on your graphs.

In programming, especially in BASIC, we should not concern ourselves with what should be, but only with what works.

SORT
elementary data processing

Computers may not be good at noticing unexpected relationships, but they are superb at working with lists. A lot of paperwork consists of taking a list organized in one way and reshuffling it. In this chapter, you will write a program that accepts lists of random data and puts them into alphabetical order.

The programmer will teach the computer to remember, sort, display, save, retrieve, and print data, useful skills for any computer.

SORT is a simple program for alphabetizing a disorganized list. The SORT routine itself is short and easily understood, and can be used in other programs. In fact, it is used again in the hot-rod version of SORT at the end of this section, and in DATEBOOK and ORGANIZE in the final section of the book.

SORT is menu-driven—you are presented with a list (a menu) of options you can choose. You will usually start by choosing the first option, ENTER DATA, because the program doesn't do anything very interesting until it has some data to sort. You enter data one piece at a time (a single piece of data is called a datum) until you're done. When you've entered all the data you want to sort, you use the computer's escape code, ZZZ, and the menu reappears.

Once the computer has some data to play with, you can choose option 3 to DISPLAY your entries, or option 2 to SORT them. If you choose to SORT, your data will flash by as it is manipulated by the computer. This program assumes you won't have hundreds of pices of data to sort, and so it uses a simple, inefficient sorting technique.

SuperSORT, the upgraded version of the sorting program, adds several more options: you can SAVE, LOAD, PRINT, and CLEAR your data.

WRITING THE PROGRAM

If you have been working through this book a section at a time, there will be nothing very surprising in this program. It breaks up easily into the MENU/Control module, the SORT module, and the DISPLAY module.

The MENU module:

```
1 REM ************************* SORT *
2 DIM A$(4000),D$(4000),I$(1),B$(40),C
$(40),M$(40),Q$(1),T$(40)
3 DIM L(100),M(100),S(100)
4 OPEN #1,4,0,"K:"
6 PRINT CHR$(125):N=1:LD=40
8 FOR N=1 TO LD:M$(LEN(M$)+1)=" ":NEXT
 N:N=1
10 PRINT ,"SORT"
11 PRINT "1: ENTER DATA"
12 PRINT "2: SORT"
13 PRINT "3: DISPLAY"
19 PRINT "9: QUIT"
20 PRINT ,"OPTION";:INPUT Q
30 ON Q GOTO 100,200,300,10,10,10,10,1
0,900
40 GOTO 10
```

This section reminds you of your choices, and sends program control off to the other modules when you have chosen.

The ENTER DATA section uses another version of the keyboard subroutine:

```
100 PRINT "ENTER DATA         ZZZ TO Q
UIT"
110 PRINT N;":";:GOSUB 1000:PRINT
120 IF T$="ZZZ" THEN ND=N-1:GOTO 10
130 A$(1+LD*(N-1))=T$:L(N)=LEN(T$)-1
140 N=N+1:GOTO 110
```

```
900 PRINT "DONE":END
1000 T$=""
1010 I=PEEK(764):IF I=255 THEN 1010
1020 GET #1,K:POKE 764,255
1030 IF K=155 THEN RETURN
1070 I$=CHR$(K):PRINT I$;
1080 T$(LEN(T$)+1)=I$
1090 GOTO 1010
```

You can "test-drive" your program now, to see if it is getting data into the string array A$(n). Check by QUITting and asking the computer to show you your A$ like this (remember, no line numbers):

```
PRINT A$
```

The SORT module looks like this:

```
200 PRINT "SORTING":FOR N=1 TO ND:S(N)
=1:NEXT N:PL=1
210 N=1
220 IF S(N)=0 THEN N=N+1:GOTO 220
225 P=N+1:LN=N:SP=1+LD*(N-1):B$=A$(SP,
SP+L(N))
230 IF S(P)=0 THEN 260
240 SQ=1+LD*(P-1):C$=A$(SQ,SQ+L(P)):PR
INT B$,C$,:IF B$<C$ THEN 260
250 T$=B$:B$=C$:C$=T$:LN=P:PRINT "SWAP
PING";
260 PRINT :P=P+1:IF P<=ND THEN 230
270 SY=1+LD*(PL-1):D$(SY)=B$:M(PL)=LEN
(B$)-1:PL=PL+1:S(LN)=0:IF PL=ND THEN 2
80
275 GOTO 210
280 SY=1+LD*(ND-1):D$(SY)=C$:M(ND)=LEN
(C$)-1
290 PRINT " = = = DONE = = =":A$=D$:FO
R N=1 TO ND:L(N)=M(N):NEXT N
```

The real work takes place in lines 240 and 250: Line 240 decodes each datum and compares it with each of the data entered after it, and if it is less than a later entry, line 250 pops the first datum into a temporary string, pops the later datum into the earlier position, and then shuffles the former datum into the latter's position. All the PRINTing in this module is diagnostic—so you can see the work taking place inside the computer, and find out what's wrong during debugging. Programmers often insert PRINTs and STOPs to help find programming problems, then take them out when the program is running smoothly. If you take out the diagnostics—that would be parts of lines 240, 250, and 260—the routine will sort lists faster, because the computer won't have to work so hard.

The DISPLAY module is simplicity itself:

```
300 FOR N=1 TO ND
310 SF=1+LD*(N-1):PRINT N;"   ";A$(SF,S
F+L(N))
320 NEXT N
390 PRINT ,,"MENU";:INPUT Q$:GOTO 10
```

With these lines programmed in, SORT should do everything advertised.

But wouldn't it be nice if you could SAVE lists, and LOAD them, and PRINT them, and MODIFY them? Read on.

SUPERSORT

a list-organizing tool

MAKING A TOY INTO A TOOL

SuperSORT is a junior version of the most important data-management tool known to man or computer: the systematic list-reorganization utility. When you get down to it, much of the work computers do centers on taking a list of data in one order—alphabetical, perhaps, so that the subscription clerk at the local newspaper can make sure names, address, and most important, the expiration date, are all correct—into another order—a carrier-route sort for the postal folks, so that the newspapers can be distributed efficiently to the subscribers.

In addition to ENTERing, DISPLAYing, SORTing, and QUITting the way you can in SORT, you can now also SAVE, LOAD, PRINT, MODIFY, and CLEAR your data. In other words, you can do all the things one generally needs to do with data. To do it well, you will need a disk drive.

UPGRADING THE PROGRAM

The menu now looks like this:

```
1 REM ****************************SSORT *
2 DIM A$(4000),D$(4000),I$(1),B$(40),C
$(40),M$(40),Q$(1),T$(40),E$(8),F$(14)
,CL$(1)
3 DIM L(100),M(100),S(100)
4 OPEN #1,4,0,"K:"
6 CL$=CHR$(125):PRINT CL$
9 N=1:LD=40:GOTO 800
```

```
10 PRINT ,"    SORT"
11 PRINT "1: ENTER DATA"
12 PRINT "2: SORT          6: PRINT"
13 PRINT "3: DISPLAY       7: EDIT"
14 PRINT "4: SAVE          8: CLEAR"
15 PRINT "5: LOAD          9: QUIT"
20 PRINT ,,"OPTION";:INPUT Q
30 ON Q GOTO 100,200,300,400,500,600,7
00,800,900
40 GOTO 10
```

The lines between 200 and 390, the ENTER, SORT, and DISPLAY modules, are exactly as in SORT.

The SAVE and LOAD modules can sensibly be entered together:

```
400 PRINT ,"FILENAME FOR SAVE";:INPUT
E$
410 F$="D:":F$(3)=E$:F$(LEN(F$)+1)=".D
TA":OPEN #2,8,0,F$
420 FOR N=1 TO ND
430 SP=1+LD*(N-1):T$=A$(SP,SP+L(N)):PR
INT #2;T$:PRINT ">";T$
440 NEXT N:PRINT #2;"ZZZ":CLOSE #2
490 GOTO 10
500 PRINT ,"FILENAME TO LOAD";:INPUT E
$
510 F$="D:":F$(3)=E$:F$(LEN(F$)+1)=".D
TA":OPEN #2,4,0,F$
520 INPUT #2,T$:IF T$="ZZZ" THEN 590
530 SP=1+LD*(N-1):A$(SP)=T$:L(N)=LEN(T
$)-1
535 PRINT "<";T$
540 N=N+1:GOTO 520
590 ND=N-1:CLOSE #2:GOTO 10
```

Lines 410 and 510 inform the intelligence within the disk drive what kind of file to expect. The program uses buffer number two, and F$ tells it the name of the file. The other numbers in the OPEN statement contain details for the machine—the 8 in line

410 prepares to write to the disk, while the 8 in line 510 prepares to read from the disk. With that information established, the program can then assume that the storage device will save anything PRINTed to buffer #2, or will provide reliable data when asked to INPUT from buffer #2. (There is no magic to the number 2, use any reasonable number. If it's unreasonable, be assured that the computer's BASIC interpreter will complain.)

The SAVE module writes (line 440) a final word, ZZZ, after your last datum but before closing the file, to mark the end of the file. The READ module reads from the disk until it encounters the end-of-file marker ZZZ.

To convert this program for tape input/output, please consult your ATARI manuals.

PRINTing your list requires this code:

```
600 PRINT "PRINT LINE NUMBERS (Y/N)";:
INPUT Q$
610 FOR N=1 TO ND
620 IF Q$<>"Y" THEN 640
630 SP=1+LD*(N-1):LPRINT N;"  ";A$(SP,
SP+L(N)):GOTO 650
640 SP=1+LD*(N-1):LPRINT A$(SP,SP+L(N)
)
650 NEXT N
690 GOTO 10
```

You can print your list with or without line numbers, like this:

with line numbers	without line numbers
1 RANGE	YOLK
2 WORLD	GUESS
3 SWAP	calf
4 PINT	does
5 EXTRA	elf
6 FIRES	extra
7 YOLK	field
8 GUESS	fired

Clearing data takes these lines:

```
800 PRINT "CLEARING MY MIND ..."
810 FOR Z=1 TO 1000:A$(Z)=" ":NEXT Z
820 FOR Z=1 TO 100:S(Z)=0:N=1
890 PRINT CL$:GOTO 10
900 PRINT "DONE":END
```

Modifying a datum is only slightly trickier.

```
700 PRINT ,"LINE TO CORRECT";:INPUT X:
IF X=0 THEN 10
710 SP=1+LD*(X-1):PRINT X;" ";A$(SP,SP
+L(X))
720 PRINT "  ";:GOSUB 1000:PRINT
730 IF T$="ZZZ" THEN 10
740 IF T$="" THEN 700
750 D$="":IF X<ND THEN D$=A$(SP+LD,LEN
(A$))
760 A$(SP)=T$:L(X)=LEN(T$)-1:A$(SP+LD)
=D$
790 GOTO 700
```

After specifying the line to correct (700), you type the correct line. Much like the ENTRY mode, you can keep correcting data until you select item 0 or type ZZZ. Let's hope none of you ever really needs a datum ZZZ!

You should be testing each module after you add it. This is a valuable program, so don't forget to save it.

ADVANCED TOPICS

Like all the programs in the book, superSORT is meant to be only a starting place as you move toward customizing your computer to do the work you need to do. If part of the program doesn't perform to your needs, fix it! For example, if you need a margin on your printed work, lengthen the bunch of spaces in line 630, or include them in line 640.

Borrow from this program, and modify it without fear, especially if you have a working copy saved on disk!

If this program suggests useful work to you, check the two adaptations in the last section—DATEBOOK and ORGANIZE. They work with three kinds of data at a time, instead of just one.

CALC
a smart calculator

The way people think, and the way machines think, are sometimes far apart. The way we write

```
2 + 3 =
```

is a good example. A computer is much happier if we tell it all the players, then tell it what game to play, like this:

```
2 3 + . *
```

But computers are supposed to be our servants, not the other way around, so programmers have to teach them to think like people! This program teaches the computer to behave like a four-function calculator with a simple memory. It uses the BASIC symbols: * means times and / means divided by.

```
CALC:2+3-4 = 1
CALC:16*24 = 384
CALC:A/12=32 YES
CALC:2+2=5 NO
CALC:
```

CALC turns your computer into a very simple "expert system." Its expertness is just like a calculator: it's good with numbers.

*One computer language, FORTH, does exactly that. If BASIC feels clunky to you, you aren't alone: many scientists and computer specialists program exclusively in FORTH. Check into it.

You can type in complex problems using the number keys, + for plus, - for minus, * for times, and / for divided by. Like most calculators, the order of the calculations is critical:

```
3 * 5 + 2
```

is not the same thing as

```
2 + * 3
```

You can use your last answer in a calculation, like this: If you calculate

```
2 + 5
```

the computer figures that out as 7. If you then ask for

```
4 * A
```

the computer substitutes your last answer, 7, for A, and informs you that the result is 28. This turns out to be a useful trait.

One more wrinkle: If you want to know if one of your calculations is right, but don't want to know the answer if you're wrong, you can type your questions like this:

```
4 * 7 = 28
```

the computer responds with a "YES" if you are right, and a "NO" if you aren't. That way you can check your homework, and learn the lessons you are supposed to, without cheating.

WRITING THE PROGRAM

The program works by accepting a whole problem, then "parsing it"—looking at it a character at a time, and planning a solution.

The keyboard is so simple that we'll throw in the program title line, too:

```
1 REM * CALC *
2 DIM A$(1ØØ),I$(1),V$(1ØØ),O(1ØØ),V(1
ØØ)
1Ø MO=Ø:PRINT "CALC";:INPUT A$
```

The "Parser" looks at each consecutive character:

```
20 LA=LEN(A$):N=1:P=1
30 I$=A$(P,P)
40 IF I$="+" THEN O(N)=1:GOTO 180
50 IF I$="-" THEN O(N)=2:GOTO 180
60 IF I$="*" THEN O(N)=3:GOTO 180
70 IF I$="/" THEN O(N)=4:GOTO 180
80 IF I$>"/" AND I$<":" THEN 170
90 IF I$="." THEN 170
100 IF I$="A" THEN 150
110 IF I$="=" THEN 130
120 GOTO 190
130 MO=1:QA=VAL(A$(P+1)):GOTO 200
150 V$=STR$(QA):GOTO 190
170 V$(LEN(V$)+1)=I$:GOTO 190
180 V(N)=VAL(V$):V$="":N=N+1
190 P=P+1:IF P<=LA THEN 30
```

When this segment is done, the problem has been broken down into values [one for each value in a variable array called V(n)] and Operations [one for each relationship between two values, stored in O(n)]. There will be one less operation, we hope, than there are values.

The next section processes the values in the manner dictated by the operations:

```
200 V(N)=VAL(V$):V$="":LN=N:N=1
210 IF LN=1 THEN 400
220 IF O(N)=1 THEN V(N)=V(N)+V(N+1):GO
SUB 300:N=1:GOTO 210
230 IF O(N)=2 THEN V(N)=V(N)-V(N+1):GO
SUB 300:N=1:GOTO 210
240 N=N+1:IF N<LN THEN 210
250 N=1:IF LN=1 THEN 400
260 IF O(N)=3 THEN V(N)=V(N)*V(N+1):GO
SUB 300:N=1:GOTO 250
270 IF O(N)=4 THEN V(N)=V(N)/V(N+1):GO
SUB 300:N=1:GOTO 250
```

```
280 N=N+1:IF N<LN THEN 250
290 N=1:IF LN=1 THEN 400
300 FOR M=N+1 TO LN-1
310 V(M)=V(M+1):O(M-1)=O(M)
320 NEXT M:LN=LN-1:RETURN
```

One by one, the results are accumulated into the first Value, V(1), and the number of values to be processed is reduced—in computer talk we call it decremented—by one, until there's only one left: that's the answer.

The answer is displayed next:

```
400 PRINT "";:FOR P=1 TO LA+6:PRINT ""
;:NEXT P
410 IF MO=1 THEN 500
420 OA=V(1):PRINT "=";OA:GOTO 10
500 IF QA=V(1) THEN PRINT "YES":GOTO 1
0
510 PRINT "NO":GOTO 10
```

If we want to see an answer, line 420 prints it. If we just want to know if we are right, lines 500 and 510 tell us. Then the program returns for another problem.

ADVANCED TOPICS

There is lots of room to add the <UP-ARROW> and exponentiation to our calculator. Two other useful abilities would be: processing ((paren)theses) and multiple memories. I leave this challenge to the advanced programmers.

If you start getting good using it, you can graduate to the computer's own "immediate mode" calculator, wherein you type in problems like this:

```
? 2 + 3 <RETURN>
? ((32.15/48.65)/12.5+3.2)+(121.9*.03)
```

and the computer provides the answer. The ATARI, it turns out, is a very sophisticated calculator.

ANYBASE
a counting machine for any planet

When they taught most of us to count, they forgot to mention that the decimal system is only one of a whole galaxy of perfectly lovely counting systems. We count with ten fingers, but computers like the ATARI, for instance, count with eight hands, each containing only one finger! And in Base 12, the Baker's Base, 10 is evenly divisible by 2, 3, 4, and 6!

ANYBASE contains two identical and interlocking counting machines. You can tell one to run as if it has ten fingers, the decimal system we use, and the second that it has only one finger, like the binary system that computers use, and watch the counters work together.

NOTE: If you have no notion why anyone would want to do that, you are not required to read any more of this chapter. On the other hand, if you know *exactly* what I'm talking about, skip a page. On the third hand, if you are interested (or not quite sure) read on:

As I was saying: on the planet, ZamoGram, all trade is conducted in base 4. You count like this:

Earth	*ZamoGram*
1	1
2	2
3	3

So far, so good. But now we get into trouble.

4	10
5	11
6	12
7	13
8	20

Are you with me? The next number after 33 would be 100, right?

If your business requires frequent translations between the ZamoGrams and the Twist-tyes (who use base base 35—you, for example, my friend, might weigh between 1J and 2Z pounds), you, I say, NEED THIS PROGRAM!

WRITING THE PROGRAM

This program is exceptionally modular: two of its modules are nearly identical. But first, the "human interface":

```
1 REM * ANYBASE *
2 DIM Q$(9),C$(9),D$(9),N(9),M(9)
3 FOR N=Ø TO 9:N(N)=Ø:M(N)=Ø:NEXT N
1Ø PRINT "      NUMBER";:INPUT Q$
2Ø PRINT "    ITS BASE";:INPUT B2
3Ø PRINT "TARGET BASE";:INPUT B1
7Ø PRINT "";:GOSUB 3ØØ:GOSUB 2ØØ
9Ø IF C$=Q$ THEN PRINT "":RUN
1ØØ GOTO 7Ø
```

Line 10 gets your desired number as a string of characters, Q$, while lines 20 and 30 get the number-base of your desired number, and the base to translate into. Line 70 sends control to the two counters. If the first counter has reached the target number, the count stops; otherwise, it loops back to line 70.

The counters look like this:

```
2ØØ N=Ø
21Ø N(N)=N(N)+1:IF N(N)<B1 THEN 23Ø
22Ø N(N)=Ø:N=N+1:GOTO 21Ø
23Ø IF N>HN THEN HN=N
24Ø FOR PP=1 TO 16:PRINT "";:NEXT PP:P
RINT ">";:FOR D=HN TO Ø STEP -1
25Ø IF N(D)<1Ø THEN D$=STR$(N(D)):GOTO
 27Ø
26Ø D$=CHR$(55+N(D))
27Ø PRINT D$;:NEXT D
29Ø PRINT :RETURN
3ØØ M=Ø:C$=""
31Ø M(M)=M(M)+1:IF M(M)<B2 THEN 33Ø
32Ø M(M)=Ø:M=M+1:GOTO 31Ø
33Ø IF M>HM THEN HM=M
```

```
340 FOR PP=1 TO 16:PRINT " ";:NEXT PP:P
RINT ">";:FOR D=HM TO 0 STEP -1
350 IF M(D)<10 THEN D$=STR$(M(D)):GOTO
 370
360 D$=CHR$(55+M(D))
370 PRINT D$;:C$(LEN(C$)+1)=D$:NEXT D
390 PRINT :RETURN
```

Line 200 (and 310—the two are parallel) starts (the fancy computer term is initializes) the place counter. Line 210 checks the current place to see if adding one to it will be less than the number-base. (The basic rule of number-bases is that there is no numeral Z in base Z.) If not, the program "falls through" to line 220, which sets the current digit to 0, shifts its attention one place higher, and takes the unexpected step of looping back on itself: recursion in a BASIC program.

If the incremented number will be less than the number-base—or when the recursive program finally resolves itself, finding or creating a place it can increment—the string of digits representing the number is adjusted and displayed by lines 240 through 270. The counter then returns to the command immediately following where it was called.

ATARI notes

Duplicating program lines revisited

ANYBASE provides a made-to-order opportunity to perfect your line-duplicating skills. If you type in lines 200 to 290, you have done 90% of the work.

If you just finished typing lines 200 to 290, they are displayed on the screen before you, and entered in program memory, as well. If you doubt (or if you have a scrambled screen), LIST 200-290.

Move the cursor with the <SHIFT><CURSOR-UP> key until you are on line 200. Press the <3> key to change the line number, then <CURSOR-RIGHT∓ over until you are on top of the N, and overstrike it with an ≅M∓. <M>. Position the cursor at the end of the line and type in the rest of line 300. Now press <RETURN>. With a few kesytrokes, you have modified and borrowed the line, with efficiency and a much-reduced chance of typing errors. This feature is called "full-screen editing," and it certainly the programmer's task easier, a tool worth mastering.

Repeat the process, making the necessary changes and accepting each corrected line with a <RETURN> as you go. Remember, if you accept a line, then notice it needs another change, you can always move your cursor back into the line, fix it, and reaccept it. The most recent acceptance always prevails.

ADVANCED SUBJECTS

There are much more efficient interbase translation algorithms. The most elegant is based on the fact that exponents of the number bases come into play here. The decimal number 3179, we all agree, is made up of

```
    9 * 1 =      9(9 units)
   7 * 10 =     70(7 tens)
  1 * 100 =    100(1 hundred)
 3 * 1000 =   3000(3 thousands)
            +
              ____
 add up to    3179
```

What you may not know is that exponents are at play here:

```
           1000   =   10^3  =  10*10*10
            100   =   10^2  =  10*10
             10   =   10^1  =  10
              1   =   10^0  =  1
 { all numbers to the zeroth power = 1 }
```

The same thing happens in binary—base 2. The binary number 1111 can be evaluated like this

```
                       2^3 =  8  =  2*2*2
                       2^2 =  4  =  2*2
                       2^1 =  2  =  2
                       2^0 =  1  =  1
                             +__
plainly adds up to            15
```

UTILITIES

for school and work

By now you know that computers can help with your work. This chapter helps you develop programs that can help you organize the events and data in your life. When you have finished these final pages, you can consider yourself an intermediate-grade BASIC-language computer programmer—with a fist full of modules to use on a variety of problems, and enough mastery over your machine to take on any problem it can solve.

DATEBOOK and ORGANIZE are simple data-base management systems. Both work with data that comes in three pieces—in DATEBOOK, these fields are the date, the event, and a note about it:

```
10/31   HALLOWEEN      PARTY @ CHAD'S
11/04   PAPER          ENGLISH-CHAUCER
11/10   TEST           SCIENCE
11/11   HOLIDAY        BAKERY
```

Each of these data pieces is called a field, and together form a record. These two programs let you reorganize data by any one, two, or three fields, so that you can view your data in the best possible order. For me, these two programs are the most useful in the book.

```
                                   Michael Potts
                                 29 October 1983
```

HEADER pops a credit, date, and title line in the upper right-hand corner of your printed work. Nothing new and exciting here, but handy. It's built so you can add it to anything you want printed out.

DATEBOOK

The computer can be a willing and accurate helper with its powerful memory and easy-to-program study aids. The programs in this section are more complicated; more useful, too.

DATEBOOK helps you prepare for the inevitable: if you type in your homework assignments (and your parties)—date, event, and description—this program helps you track them, so you need never be unprepared.

DATEBOOK is menu-driven: at any point your options are displayed for you. They are:

```
                 1: ENTER DATA
      2: SORT            6: PRINT
      3: DISPLAY         7: MODIFY DATA
      4: SAVE            8: CLEAR
      5: LOAD            9:QUIT
```

If you select option 1, you are prompted (sample responses in italics):

DATE/TIME:*12/25*
EVENT:*Christmas*
NOTE:*special dinner in Santiago*

If you ask for a display or a print-out of your calendar, you will be asked for the field number (DATE is field 1, NOTE is field 3) and the text you wish to match. If you had entered all your paper assignments through the end of the semester, for example, along with parties, birthdays, games, and other important dates, you could pop a list of due dates in order to plan your time.

WRITING THE PROGRAM

Those who have already programmed SSORT recognized the Menu display, and you are right: you won't have to work very hard to get DATEBOOK running.

(For those of you who want a blow-by-blow explanation of what the program does, the exciting part is back in the section on SORT.)

Initialization and Menu:

```
1 REM ********************** DATEBOOK *
2 DIM A$(4000),D$(4000),I$(1),B$(76),C
$(76),M$(76),Q$(1),T$(76),E$(76),F$(76
),N$(76),CL$(1)
3 DIM L(100,2),M(100,2),S(100)
4 OPEN #1,4,0,"K:"
6 CL$=CHR$(125):PRINT CL$
9 N=1:LD=76:L1=14:L2=39:REM GOTO 800
10 PRINT ,"   DATEBOOK"
11 PRINT "1: ENTER DATA"
12 PRINT "2: SORT          6: PRINT"
13 PRINT "3: DISPLAY       7: EDIT"
14 PRINT "4: SAVE          8: CLEAR"
15 PRINT "5: LOAD          9: QUIT"
20 PRINT ,,"OPTION";:INPUT Q
30 ON Q GOTO 100,200,300,400,500,600,7
00,800,900
40 GOTO 10
```

Data entry:

```
100 PRINT "ENTER DATA            ZZZ TO Q
UIT"
110 PRINT "DATE/TIME:";:GOSUB 1000:PRI
NT
120 IF T$="ZZZ" THEN ND=N-1:GOTO 10
130 D$=T$:PRINT "     EVENT:";:GOSUB 10
00:PRINT
140 E$=T$:PRINT "      NOTE:";:GOSUB 10
00:PRINT
150 N$=T$:GOSUB 1100
180 A$(1+LD*(N-1))=T$
190 N=N+1:GOTO 110
```

Sort module:

```
200 PRINT "SORTING":FOR N=1 TO ND:S(N)
=1:NEXT N:PL=1
210 N=1
220 IF S(N)=0 THEN N=N+1:GOTO 220
225 P=N+1:LN=N:SP=1+LD*(N-1):B$=A$(SP,
SP+L2+L(N,2)-1)
230 IF S(P)=0 THEN 260
240 HN=P:SQ=1+LD*(P-1):C$=A$(SQ,SQ+L2+
L(P,2)-1):IF B$<C$ THEN 260
250 T$=B$:B$=C$:C$=T$:LN=P:HN=N
260 P=P+1:IF P<=ND THEN 230
270 SY=1+LD*(PL-1):D$(SY)=B$:FOR ZL=0
TO 2:M(PL,ZL)=L(LN,ZL):NEXT ZL:PL=PL+1
:S(LN)=0:IF PL=ND THEN 280
275 GOTO 210
280 SY=1+LD*(ND-1):D$(SY)=C$:FOR ZL=0
TO 2:M(ND,ZL)=L(HN,ZL):NEXT ZL
290 PRINT " = = = DONE = = =":A$=D$
295 FOR N=1 TO ND:FOR ZL=0 TO 2:L(N,ZL
)=M(N,ZL):NEXT ZL:NEXT N:GOTO 310
```

Display:

```
300 GOSUB 1400:LM=LEN(M$)
310 FOR N=1 TO ND:SP=1+LD*(N-1):T$=A$(
SP,SP+L2+L(N,2)-1)
320 GOSUB 1150:IF LM=0 THEN 370
330 ON FM GOTO 340,350,360
340 IF LM>LEN(D$) THEN 380
343 IF M$=D$(1,LM) THEN 370
346 GOTO 380
350 IF LM>LEN(E$) THEN 380
353 IF M$=E$(1,LM) THEN 370
356 GOTO 380
360 IF LM>LEN(N$) THEN 380
363 IF M$=N$(1,LM) THEN 370
366 GOTO 380
370 PRINT D$,E$:PRINT "   ";N$
380 NEXT N
390 PRINT ,,"MENU";:INPUT Q$:GOTO 10
```

Save and Load:

```
400 PRINT ,"FILENAME FOR SAVE";:INPUT
E$
410 F$="D:":F$(3)=E$:F$(LEN(F$)+1)=".D
TA":OPEN #2,8,0,F$
420 FOR N=1 TO ND
430 SP=1+LD*(N-1):T$=A$(SP,SP+L2+L(N,2
)-1):GOSUB 1150:IF D$="" THEN 480
440 PRINT #2;D$:PRINT #2;E$:PRINT #2;N
$:PRINT ">";D$,E$
480 NEXT N:PRINT #2;"ZZZ"
490 CLOSE #2:GOTO 10
500 PRINT ,"FILENAME TO LOAD";:INPUT E
$
510 F$="D:":F$(3)=E$:F$(LEN(F$)+1)=".D
TA":OPEN #2,4,0,F$
520 INPUT #2,D$:IF D$="ZZZ" THEN 590
530 INPUT #2,E$:INPUT #2,N$
540 GOSUB 1100:SP=1+LD*(N-1):A$(SP)=T$
545 PRINT "<";D$,E$
550 N=N+1:GOTO 520
590 ND=N-1:CLOSE #2:GOTO 10
```

Print module:

```
600 GOSUB 1400:LM=LEN(M$)
610 FOR N=1 TO ND:SP=1+LD*(N-1):T$=A$(
SP,SP+L2+L(N,2)-1)
620 GOSUB 1150:IF LM=0 THEN 670
630 ON FM GOTO 640,650,660
640 IF LM>LEN(D$) THEN 680
643 IF M$=D$(1,LM) THEN 670
646 GOTO 680
650 IF LM>LEN(E$) THEN 680
653 IF M$=E$(1,LM) THEN 670
656 GOTO 680
660 IF LM>LEN(N$) THEN 680
663 IF M$=N$(1,LM) THEN 670
666 GOTO 680
670 LPRINT D$,E$,N$
680 NEXT N
690 GOTO 10
```

Reschedule:

```
700 GOSUB 1400:LM=LEN(M$):IF FM=0 THEN
 10
710 FOR N=1 TO ND:SP=1+LD*(N-1):T$=A$(
SP,SP+L2+L(N,2)-1)
720 GOSUB 1150:IF LM=0 THEN 770
730 ON FM GOTO 740,750,760
740 IF LM>LEN(D$) THEN 790
743 IF M$=D$(1,LM) THEN 770
746 GOTO 790
750 IF LM>LEN(E$) THEN 790
753 IF M$=E$(1,LM) THEN 770
756 GOTO 790
760 IF LM>LEN(N$) THEN 790
763 IF M$=N$(1,LM) THEN 770
766 GOTO 790
770 GOSUB 1700:IF T$="ZZZ" THEN N=ND+1
:GOTO 10
780 D$="":IF N=ND THEN 785
783 D$=A$(SP+LD,LEN(A$))
785 A$(SP)=T$:A$(SP+LD)=D$
790 NEXT N:GOTO 700
800 PRINT "CLEARING MY MIND ..."
810 FOR Z=1 TO 1000:A$(Z)=" ":NEXT Z
820 FOR Z=1 TO 100:S(Z)=0:NEXT Z
890 N=1:PRINT CL$:GOTO 10
```

End and the Keyboard:

```
900 PRINT "DONE":END
1000 T$=""
1010 I=PEEK(764):IF I=255 THEN 1010
1020 GET #1,K:POKE 764,255
1030 IF K=155 THEN RETURN
1070 I$=CHR$(K):PRINT I$;
1080 T$(LEN(T$)+1)=I$
1090 GOTO 1010
```

Taking strings apart and putting them together:

```
1100 T$=D$:T$(L1)=E$:T$(L2)=N$
1110 L(N,0)=LEN(D$)-1:L(N,1)=LEN(E$)-1
:L(N,2)=LEN(N$)-1
1140 RETURN
1150 D$="":IF L(N,0)<0 THEN 1160
1155 D$=T$(1,1+L(N,0))
1160 E$="":IF L(N,1)<0 THEN 1170
1165 E$=T$(L1,L1+L(N,1))
1170 N$="":IF L(N,2)<0 THEN 1190
1180 N$=T$(L2,L2+L(N,2))
1190 RETURN
```

Matching fields and data:

```
1400 M$="":PRINT "FIELD TO MATCH";
1410 INPUT F$:IF F$="" THEN FM=0:GOTO
1490
1420 FM=VAL(F$):PRINT " DATA TO MATCH"
;
1430 INPUT M$
1490 RETURN
```

Data modifier:

```
1700 PRINT D$,E$:PRINT "   ";N$
1710 PRINT "DATE/TIME:";:GOSUB 1000:PR
INT
1720 IF T$="ZZZ" THEN RETURN
1740 D$=T$
1750 PRINT "     EVENT:";:GOSUB 1000:PR
INT :IF T$="" THEN 1770
1760 E$=T$
1770 PRINT "      NOTE:";:GOSUB 1000:PR
INT :IF T$="" THEN 1790
1780 N$=T$
1790 GOSUB 1100:RETURN
```

USING DATEBOOK

There are some tricks that may help you use DATEBOOK more effectively. You will notice that sorting sometimes gives unexpected results. For example,

```
9/30
```

the usual abbreviation for September 30th, will sort after

```
10/13
```

because the computer thinks 10/ is less than 9/3—and it's right. (Start counting at the leftmost character: 1 is less than 9, isn't it?)

You can avoid that problem by using a leading 0, like

```
09/30
```

Erase the record from the calendar by rescheduling the event's DATA to a blank—press <RETURN> only. You can reschedule an event by changing the DATE, then resorting the data.

You can load lists onto the ends of lists, and save them as one giant list simply by loading more than one file.

ORGANIZE: TO DO FOR DATA WHAT YOU DID FOR DATES

ORGANIZE lets you work with other kinds of facts the way DATEBOOK keeps your calendar: it's a special-purpose data base manager to customize for any kind of need you can imagine.

WRITING THE PROGRAM

Upgrading DATEBOOK takes a few easy changes.

Change the program name in line 1 and 10 (although this is optional, it will cut down on confusion later.)

Change the prompts in lines 110, 130, 140, 1710, 1750, and 1770.

There is a full listing of DATEBOOK in the appendix for comparison and debugging.

You are finished: ORGANIZE is ready for testing and use.

HEADER

You can plug HEADER in to a program that you are using for school or work to put a name, date, and title on every page, like this:

```
                                   Michael Potts
                                   6 November 1983
                                   Heading Program
```

WRITING THE PROGRAM

There isn't much to it (but it makes your work look good):

```
2 DIM D$(16),T$(16),TB$(55)
6000 PRINT "today's date";:INPUT D$
6010 PRINT "       title";:INPUT T$
6020 FOR N=1 TO 55:TB$(N)=" ":NEXT N
6030 LPRINT TB$;"Michael Potts"
6040 LPRINT TB$;D$
6050 LPRINT TB$;T$
```

You insert a subroutine call (GOSUB 6000) right at the beginning of a print routine. A bit of attention to your printer—always start typing at the same place on a clean sheet—and you have nearly professional output.

appendix

Programming is one activity that uses the computer usefully, but there are other interesting things to do.

Word processing turns your computer into a very smart typewriter. I can help with organizing and writing papers, letters, and other written work. You could write or buy utilities that make your computer even smarter for writing: spelling checkers, table-of-contents generators, fancy-printing programs. The limitation is your time (or budget) and imagination. Atari Writer and Letter Perfect are two word processors for the ATARI.

Spread-sheet programs turn your computer into an excellent accountant's helper—a sort of super calculating machine. With a spreadsheet program loaded into your computer, you can do elaborate simulations and "What-if" evaluations, or simply balance your personal finances. Visi-Calc is such a program.

You need keyboard skills to use a computer well—knowing which finger hits which key. Good computerized touch-typing instruction courses and games exist that will make your dialogue with the computer much more efficient and enjoyable.

The outside world is available to your computer, and with a modem and communications software you can take control of some of the most powerful computers in the world. For more information on this, an excellent resource is the book The Complete Handbook of Personal Computer Communications (Alfred Glossbrenner, St. Martin's Press, New York, 1983.) The Source, CompuServe, and Dialog are three of the most interesting computers for your computer to call.

NUMS

```
1 REM ************************ NUMS *
2 DIM A$(100),B$(3),N$(20),R$(1)
3 PRINT CHR$(125)
4 OPEN #1,4,0,"K:"
5 DATA 1,2,3,4,5,6,7,8,9,0,END
10 PRINT :PRINT "HELLO.":PRINT "MY NAM
E IS NUMS."
20 PRINT "WHAT IS YOUR NAME";:INPUT N$
:N=1
30 READ B$:IF B$="END" THEN 50
40 A$(N)=B$:N=N+1:GOTO 30
50 LN=1:HN=N-1
60 GOSUB 1300:REM GETS A RANDOM NUMBER
70 GRAPHICS 2+16:POSITION 10,5:R$=A$(R
,R):PRINT #6;R$;
80 GOSUB 1000:B$=CHR$(K):REM GETS KEY
90 IF B$=R$ THEN 110
100 GOTO 80
110 POSITION 8,6:HN=5:LN=1
120 GOSUB 300:PRINT #6;CHR$(89+R*32);
130 GOSUB 300:PRINT #6;CHR$(69+R*32);
140 GOSUB 300:PRINT #6;CHR$(83+R*32);
150 PRINT #6;" !"
200 FOR T=1 TO 800:NEXT T
290 GOTO 50
300 GOSUB 1300:IF R=2 THEN 300
310 IF R=3 THEN 300
390 RETURN
1000 I=PEEK(764):IF I=255 THEN 1000
1010 GET #1,K:POKE 764,255
1090 RETURN
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

COUNTEM

```
1 REM ********************** COUNTEM *
2 DIM C$(1),D$(6),M$(12),Q$(1)
3 C$=CHR$(125):M$="            "
4 OPEN #1,4,0,"K:"
5 PRINT C$;M$;"* COUNTEM *"
10 PRINT D$;"WHAT IS THE LOWEST NUMBER
";:INPUT LL
20 PRINT "         THE HIGHEST NUMBER";:
INPUT HH
30 LN=LL:HN=HH:GOSUB 1300:GRAPHICS 2+1
6:POSITION 1,5
40 FOR N=1 TO R:PRINT #6;" *";:NEXT N
60 POSITION 3,8:PRINT #6;"HOW MANY? ";
:GOSUB 1000
70 IF Q=R THEN 110
8Ø POSITION 2,4:FOR N=1 TO R:PRINT #6;
N;" ";:NEXT N
9Ø GOTO 6Ø
11Ø POSITION 8,6:HN=5:LN=1
12Ø GOSUB 3ØØ:PRINT #6;CHR$(89+R*32);
13Ø GOSUB 3ØØ:PRINT #6;CHR$(69+R*32);
14Ø GOSUB 3ØØ:PRINT #6;CHR$(83+R*32);
15Ø PRINT #6;" '"
2ØØ FOR T=1 TO 8ØØ:NEXT T
29Ø GOTO 3Ø
3ØØ GOSUB 13ØØ:IF R=2 THEN 3ØØ
31Ø IF R=3 THEN 3ØØ
39Ø RETURN
1ØØØ I=PEEK(764):IF I=255 THEN 1ØØØ
1Ø1Ø GET #1,K:POKE 764,255
1Ø2Ø Q$=CHR$(K)
1Ø3Ø IF Q$<"1" OR Q$>"9" THEN 1ØØØ
1Ø4Ø Q=VAL(Q$):PRINT #6;Q
1Ø9Ø RETURN
13ØØ R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

FINDME

```
1 REM ************************ FINDME *
2 DIM A$(1000),B$(20),D$(20),M$(12),C$
(12),R$(60),I$(1),T$(20),Q$(20)
5 C$=CHR$(125)
10 PRINT C$;"* FINDME *":PRINT "Y"
20 FOR R=8 TO 0 STEP -1:PRINT R;
30 FOR C=0 TO 9:PRINT " + ";:NEXT C:PR
INT :PRINT
40 NEXT R:FOR C=0 TO 9:PRINT "  ";C;:N
EXT C:PRINT "X"
50 HN=9:GOSUB 1300:X=R:HN=8:GOSUB 1300
:Y=R
60 VX=X:VY=Y:Z=88:GOSUB 1200
70 POSITION 1,22:PRINT "CAN YOU FIND M
E (X,Y)";
80 INPUT QX,QY:IF QX=X AND QY=Y THEN 1
00
90 GOTO 200
100 PRINT "YOU FOUND ME!";
180 FOR T=0 TO 1999:NEXT T:GOTO 10
200 PRINT "  YOU MISSED ME";
210 VX=QX:VY=QY:Z=79:GOSUB 1200
220 POSITION 1,23:PRINT " ";:FOR T=0 T
O 500:NEXT T
230 PRINT "                    ";
240 GOTO 70
1200 POSITION 4+(3*VX),2+(2*(8-VY))
1210 PRINT CHR$(Z);
1290 RETURN
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

```
1 REM ************************SFINDME *
2 DIM N(10),C$(1),M$(12)
5 C$=CHR$(125):GOSUB 1900
10 PRINT C$;M$;"* FINDME *":PRINT "Y"
20 FOR R=8 TO 0 STEP -1:PRINT R;
30 FOR C=0 TO 9:PRINT " + ";:NEXT C:PR
INT :PRINT
40 NEXT R:FOR C=0 TO 9:PRINT "  ";C;:N
EXT C:PRINT "X"
50 HN=9:GOSUB 1300:X=R:HN=8:GOSUB 1300
:Y=R
60 N1=N(X):N2=N(Y):GOSUB 2000
70 POSITION 1,22:PRINT "CAN YOU FIND M
E (X,Y)";
80 INPUT QX,QY:IF QX=X AND QY=Y THEN 1
00
90 GOTO 200
100 PRINT "YOU FOUND ME!";
110 N1=N(X):N2=N1:GOSUB 2000
120 N1=N(Y):N2=N1:GOSUB 2000
130 N1=N(3):N2=N(5):GOSUB 2000:GOSUB 2
000
140 N1=N(7):N2=N(8):GOSUB 2000
180 FOR T=0 TO 499:NEXT T
190 GOTO 10
200 PRINT "  YOU MISSED ME";
210 N1=N(QX):N2=N(X):GOSUB 2000
220 N1=N(QY):N2=N(Y):GOSUB 2000
270 POSITION 1,23:PRINT "  ";
280 PRINT "                    ";
290 GOTO 70
900 DATA 121,108,96,81,72,60,53,47,40,
35
```

```
1200 POSITION 4+(3*VX),2+(2*(8-VY))
1210 PRINT CHR$(Z);
1290 RETURN
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN

1900 FOR N=0 TO 9:READ NN:N(N)=NN:NEXT
 N
1910 N1=121:N2=81:REM *****TWONOTE*
1920 GOSUB 2000:GOTO 1910
2000 SOUND 0,N1,10,8
2010 FOR T=1 TO 40:NEXT T
2020 SOUND 0,N1,10,0
2030 FOR T=1 TO 4:NEXT T
2040 SOUND 0,N2,10,7
2050 FOR T=1 TO 80:NEXT T
2090 SOUND 0,N2,10,0:RETURN
```

ONECLAP

```
1 REM ********************** ONECLAP *
2 DIM N$(20),MO$(10),A$(4000),B$(1000)
,T$(40),F$(4),W$(4),I$(1),Q$(1),V$(5),
C$(25)
3 DIM P(1000),W(5)
4 OPEN #1,4,0,"K:"
7 V$="AEIOU":RT=300
10 C$=CHR$(125):C$(2)="             *
ONECLAP *":PRINT C$:MO$="ANY KEY":MO=0
20 PRINT :PRINT "WHAT IS YOUR NAME";:I
NPUT N$:GOTO 3000
30 GOSUB 300:IF LEN(T$)>1 THEN 55
40 GOSUB 400:G=G+1:IF G<10 THEN 30
50 MO$="ONE CLAP":MO=1:GOSUB 300
55 LT=LEN(T$):IF LT<3 THEN 250
60 F$=T$(LT-1,LT-1):GOSUB 500:IF F=-1
THEN 250
70 W=0:NW=W(F+1)-W(F)
80 WP=W(F)+W:W$=A$(P(WP)+1,P(WP+1)):IF
 T$=W$ THEN 100
```

```
85 W=W+1:IF W<NW THEN 80
90 GOTO 250
100 POSITION 3,6:PRINT #6;"IS ON MY LI
ST.":GOSUB 800
110 NR=NR+1:IF T$="QUIT" THEN 700
190 GOTO 50
250 POSITION 1,8:PRINT #6;"IS NOT ON M
Y LIST.":GOSUB 800
260 B$(LEN(B$)+1)=" ":B$(LEN(B$)+1)=T$
290 GOTO 50
300 GRAPHICS 2+16:PRINT #6;MO$
310 POSITION 8,5:PRINT #6;"?";:POSITIO
N 8,5
320 T$="":T=0:GOSUB 1000
390 RETURN
400 F$=T$:GOSUB 500:IF F<0 THEN 420
410 HN=W(F+1)-W(F):GOSUB 1300:WP=W(F)+
R:W$=A$(P(WP)+1,P(WP+1)):GOTO 480
420 HN=4:GOSUB 1300:W=0:NW=W(R+1)-W(R)
425 NW=W(R+1)-W(R)
430 WP=W(R)+W:W$=A$(P(WP)+1,P(WP+1)):I
F T$=W$(1,1) THEN 480
440 IF T$=W$(LEN(W$)) THEN 480
450 W=W+1:IF W<NW THEN 430
460 R=R+1:W=0:IF R<5 THEN 425
470 R=0:GOTO 425
480 POSITION 8,5:PRINT #6;W$
490 GOSUB 800:RETURN
500 F=-1:IF F$="A" THEN F=0
510 IF F$="E" THEN F=1
520 IF F$="I" THEN F=2
530 IF F$="O" THEN F=3
540 IF F$="U" THEN F=4
590 RETURN
700 PRINT C$:PRINT :PRINT
710 PRINT N$;" FOUND ";NR:PRINT ,"GOOD
 WORDS."
720 PRINT :PRINT "WORDS NOT ON MY LIST
:":PRINT :P=1:LB=LEN(B$)
730 PP=37:IF LB-P<37 THEN 790
740 IF B$(P+PP,P+PP)=" " THEN 760
```

```
750 PP=PP-1:GOTO 740
760 PRINT B$(P,P+PP-1):P=P+PP+1:GOTO 7
30
790 PRINT B$(P,LB):END
800 FOR T=0 TO 750:NEXT T:RETURN
1000 I=PEEK(764)
1010 T=T+1:IF T<RT THEN 1050
1020 IF MO>0 OR LEN(T$)=0 THEN 1050
1030 POSITION 6,10:PRINT #6;"PRESS [RE
TURN]";:T=0
1040 POSITION 8+LEN(T$),5
1050 IF I=255 THEN 1000
1060 GET #1,K:POKE 764,255
1070 IF K=155 THEN RETURN
1080 I$=CHR$(K):PRINT #6;I$;:T$(LEN(T$
)+1)=I$
1090 GOTO 1000
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN

3000 PRINT "EXCUSE ME,";N$
3010 PRINT "I AM READING MY LIST NOW."
3020 PRINT :W=0:LL=0:PP=0:T$=V$(1,1):V
=0:W(0)=0:P(0)=0
3030 READ W$:LW=LEN(W$):IF W$(LW-1,LW-
1)>T$ THEN 3100
3040 PRINT W$;" ";:LL=LL+LW+1:IF LL>33
 THEN PRINT :LL=0
3050 A$(PP+1)=W$:PP=PP+LW:W=W+1:P(W)=P
P
3090 GOTO 3030
3100 V=V+1:W(V)=W:IF V=5 THEN 30
3110 T$=V$(V+1,V+1):GOTO 3030
9000 DATA BAT,CAT,EAT,FAT,HAT,MAT,PAT,
RAT,SAT,TAT,VAT,WAX
9010 DATA BET,GET,JET,LET,MET,NET,PET,
SET,VET,YET
9020 DATA BIT,FIT,HIT,KIT,PIT,SIT,TIT,
WIT,ZIP,QUIT
9030 DATA COT,DOT,GOT,HOT,JOT,LOT,NOT,
POT,ROT,SOT,TOT
9040 DATA BUT,CUT,GUT,HUT,JUT,NUT,PUT,
RUT
9990 DATA ZZZ
```

PATTERN

```
1 REM ********************** PATTERNS *
2 DIM Q$(3),P$(11):P$="PATTERNS   "
5 LN=1
10 GRAPHICS 2+16:P=1:X=1:Y=1:POKE 764,
255:R=1
20 POSITION X,Y:PRINT #6;P$(P,P);:K=PE
EK(764):IF K<>255 THEN GRAPHICS 0:GOTO
 60
30 P=P+1:IF P>8+R THEN P=1
40 X=X+1:IF X<19 THEN 20
50 X=1:Y=Y+1:IF Y<10 THEN 20
55 Y=1:POSITION 1,5:PRINT #6;"< PRESS
 ANY KEY >":HN=4:GOSUB 1300:R=R-1:GOTO
 20
60 PRINT "PATTERNS":PRINT :PRINT :PRIN
T "EASY (0) OR HARD (9)";:INPUT DF
70 PRINT :HN=INT(DF/2)+1:IF HN>5 THEN
HN=5
80 GOSUB 1300:ON R GOTO 100,200,300,40
0,500
100 HN=4*(DF+1):GOSUB 1300:S=R
120 HN=2*(DF+1):GOSUB 1300:I=R
130 HN=2:GOSUB 1300:L=3+R
140 HN=L:GOSUB 1300:M=R
150 FOR N=1 TO L
160 IF N=M THEN PRINT " ? ";:GOTO 180
170 PRINT " ";S+I*(N-1);" ";
180 NEXT N:A=S+I*(M-1):GOSUB 1000:IF Q
=A THEN 1100
190 GOTO 140
200 HN=4*SQR(DF+1):GOSUB 1300:S=R
220 HN=2*(DF+1):GOSUB 1300:I=R
230 HN=2:GOSUB 1300:L=3+R
240 HN=L:GOSUB 1300:M=R
250 FOR N=1 TO L
260 IF N=M THEN PRINT " ? ";:GOTO 280
```

```
270 PRINT " ";S*I*N;" ";
280 NEXT N:A=S*I*M:GOSUB 1000:IF Q=A T
HEN 1100
290 GOTO 240
300 HN=DF:GOSUB 1300:S=R
320 HN=DF+1:GOSUB 1300:I=R
325 HN=INT(DF/2):GOSUB 1300:I2=R
330 HN=2:GOSUB 1300:L=3+R
340 HN=L:GOSUB 1300:M=R
350 I1=I:FOR N=1 TO L
360 IF N=M THEN PRINT " ? ";:A=S+I1*(N
-1):GOTO 380
370 PRINT " ";S+I1*(N-1);" ";
380 I1=I1+I2:NEXT N:GOSUB 1000:IF Q=A
THEN 1100
390 GOTO 340
400 IF FR=0 THEN 1600
410 HN=FR:GOSUB 1300:R=R-1:A$=D$(P(R)+
1,P(R+1)):P=1
420 L=LEN(A$):HN=L:GOSUB 1300:M=R
430 HN=2:GOSUB 1300:I=R:IF I=1 THEN 45
0
440 I=-1:P=L
450 IF P=M THEN PRINT " ? ";:Z$=A$(P,P
):GOTO 470
460 PRINT " ";A$(P,P);" ";
470 P=P+I:IF P=0 OR P>L THEN 490
480 GOTO 450
490 GOSUB 1000:IF Q$=Z$ THEN 1100
495 IF I<0 THEN 440
496 P=1:GOTO 450
500 HN=SQR(DF):GOSUB 1300:S=R
520 HN=2:GOSUB 1300:I=R+1
530 HN=2:GOSUB 1300:L=4+R
540 HN=L:GOSUB 1300:M=R
550 FOR N=1 TO L:Z=INT(S*N^I+0.5)
560 IF N=M THEN PRINT " ? ";:A=Z:GOTO
580
570 PRINT " ";Z;" ";
580 NEXT N:GOSUB 1000:IF Q=A THEN 1100
```

```
590 GOTO 540
1000 PRINT
1010 PRINT "WHAT IS MISSING";:INPUT Q$
1020 IF Q$="" THEN 1090
1030 IF ASC(Q$)>64 THEN 1090
1040 Q=VAL(Q$)
1090 RETURN
1100 PRINT "YOU ARE RIGHT."
1190 GOTO 70
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN

1600 DIM D$(1000),A$(20),Z$(1),P(100):
N=0
1610 READ A$:IF A$="ZZZ" THEN 1690
1620 LD=LEN(D$):P(W)=LD:D$(LD+1)=A$:W=
W+1:GOTO 1610
1690 P(W)=LEN(D$):FR=W:GOTO 410
9000 DATA BIRTHDAY,WASHINGTON,FISH,COM
PUTER,HORSE,BOAT
9990 DATA ZZZ
```

REMEMBER

```
2 DIM C$(13),T$(10),Q$(10),M$(8),R$(1)
,I$(1)
4 OPEN #1,4,0,"K:"
5 C$=CHR$(125):C$(2)="REMEMBER...  ":PR
INT C$
6 M$=""
9 TK=0.015:SK=33
10 LN=0:PRINT :PRINT :PRINT
20 PRINT "0=EASY 9=HARD  ...HOW TOUGH"
;:INPUT DF:TD=200*(11-DF)
30 L=3+(DF/3):HN=9:IF DF>5 THEN HN=36
35 IF DF>7 THEN HN=22
40 T$="":FOR N=1 TO L:GOSUB 1300
```

```
50 IF R>9 THEN 70
60 R$=STR$(R):GOTO 80
70 R$=CHR$(54+R)
80 T$(LEN(T$)+1)=R$:NEXT N
100 PRINT C$;DF;M$;:FOR N=1 TO LEN(T$)
:PRINT " ";T$(N,N);:NEXT N
110 FOR T=1 TO TD:NEXT T
120 PRINT C$;DF;M$;"?";:Q$="":T=0:POKE
 764,255
130 GOSUB 1000:T=T+1
170 IF K=155 THEN PRINT :PRINT :GOTO 2
10
180 Q$(LEN(Q$)+1)=I$:PRINT I$;" ";
190 IF Q$<>T$ THEN 130
200 PRINT :PRINT :PRINT "EXACTLY!"
210 IF Q$="" THEN 300
220 LT=LEN(T$):LQ=LEN(Q$):SC=0:FOR N=1
 TO LT
230 IF N>LQ THEN 250
240 IF T$(N,N)=Q$(N,N) THEN SC=SC+1
250 NEXT N:PRINT "SYMBOLS RIGHT:";SC,
255 PRINT INT(100*SC/LT);"%"
260 SC=INT((SK*L)/((T*TK)*(SC/LT)))+1
280 PRINT "TIME: ";T*TK;" SECONDS":PRI
NT "SCORE ";SC:TS=TS+SC:NQ=NQ+1:TT=TT+
T*TK
290 FOR T=1 TO TD:NEXT T:GOTO 40
300 PRINT C$;DF:PRINT
310 PRINT "SCORES:"
320 PRINT "TOTAL SCORE         ";TS
330 PRINT "STRINGS ATTEMPTED   ";NQ
340 PRINT "AVERAGE SCORE/$     ";TS/NQ
350 PRINT "TOTAL SECONDS       ";TT
360 PRINT "AVERAGE TIME/$      ";TT/NQ
390 FOR T=1 TO 3*TD:NEXT T:GOTO 40
1000 I=PEEK(764):T=T+1:IF I=255 THEN 1
000
1010 GET #1,K:POKE 764,255
1020 I$=CHR$(K)
1090 RETURN
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

SPELL

```
1 REM ************************ SPELL *
2 DIM A$(1ØØØ),B$(2Ø),D$(2Ø),M$(12),C$
(12),R$(6Ø),I$(1),T$(2Ø),Q$(2Ø),H$(38)
3 DIM P(3),Q(3),W(5Ø),X(5Ø),Y(25)
4 OPEN #1,4,Ø,"K:"
5 C$=CHR$(125):FOR N=2 TO 12:C$(N)="":
NEXT N:PRINT C$
6 M$="":PRINT M$;"* SPELL *"
8 P=1:FOR N=1 TO 3:READ T$:R$(P)=T$:P(
N)=P:P=LEN(R$):Q(N)=P:P=P+1:NEXT N
9 TD=1ØØØ:LN=1:NW=1:PW=1:H=Ø
1Ø GOSUB 5ØØ:REM GET DATA
2Ø FOR N=1 TO ND:PRINT A$(W(N),X(N))
3Ø IF H=Ø THEN PRINT A$(X(N)+1,Y(N)):P
RINT
4Ø NEXT N
45 H=1:GOTO 2ØØ
5Ø FOR T=1 TO TD:NEXT T
6Ø HN=ND:GOSUB 13ØØ:PRINT C$;M$;A$(W(R
),X(R))
7Ø FOR T=1 TO TD:NEXT T:PRINT C$;M$;:I
NPUT Q$
8Ø IF Q$=A$(W(R),X(R)) THEN 1ØØ
9Ø PRINT "SORRY, IT'S ";A$(W(R),X(R)):
WR=WR+1:GOTO 13Ø
1ØØ HN=3:GOSUB 13ØØ:PRINT :PRINT R$(P(
R),Q(R))
11Ø RI=RI+1
12Ø FOR T=Ø TO TD:NEXT T:IF H=1 THEN 2
ØØ
13Ø IF RI<1Ø THEN 6Ø
14Ø IF H=1 THEN 2ØØ
15Ø HN=ND:GOSUB 13ØØ:PRINT C$;"   ";A$(
X(R)+1,Y(R))
16Ø PRINT :PRINT M$;:INPUT Q$:GOTO 8Ø
```

```
200 WG=0:RG=0:HN=ND:GOSUB 1300:PRINT C
$;M$;
210 T$=A$(W(R),X(R)):LT=LEN(T$):FOR Z=
1 TO LT
220 PRINT "_ ";
230 NEXT Z:PRINT :PRINT :PRINT M$;"GUE
SS? ";
240 GOSUB 1000:Z=1:PRINT :PRINT "";M$;
250 IF I$=T$(Z,Z) THEN 310
260 PRINT "";
270 Z=Z+1:IF Z<=LT THEN 250
280 PRINT "":GOSUB 400:IF RF=1 THEN RF
=0:GOTO 240
290 WG=WG+1:IF WG<2*LT THEN 240
300 PRINT :PRINT "YOU LOSE":GOTO 130
310 PRINT I$;" ";
330 RG=RG+1:RF=1
340 IF RG=LT THEN PRINT :PRINT :GOTO 1
00
350 GOTO 270
500 PRINT :PRINT "ENTER DATA      ZZZ T
O QUIT"
510 PRINT NW;":";:INPUT T$
520 IF T$="ZZZ" THEN 590
530 A$(PW)=T$:W(NW)=PW:PW=LEN(A$):X(NW
)=PW:PW=PW+1
540 IF H=1 THEN 580
550 PRINT "HINT";:INPUT H$:IF H$="" TH
EN H=1:GOTO 580
560 GOTO 600:REM CHECK SPELLING
570 A$(PW)=H$:PW=LEN(A$):Y(NW)=PW:PW=P
W+1
580 NW=NW+1:GOTO 510
590 ND=NW-1:RETURN
600 PP=1:LT=LEN(T$)-1:LH=LEN(H$)
610 PQ=PP+LT:IF H$(PP,PQ)=T$ THEN 660
620 PP=PP+1:IF PP<=LH-LT THEN 610
630 PRINT "PLEASE CHECK SPELLING":GOTO
 550
```

```
66Ø IF FP=LH-LT THEN Q$="":GOTO 68Ø
67Ø Q$=H$(FQ+1,LH)
68Ø FOR P=FP TO FQ:H$(P)="_":NEXT P
69Ø H$(P)=Q$:GOTO 57Ø
9ØØ DATA RIGHT!!,CORRECT.,YOU GOT IT!
1ØØØ I=PEEK(764):IF I=255 THEN 1ØØØ
1Ø1Ø GET #1,K:POKE 764,255
1Ø9Ø I$=CHR$(K):RETURN
13ØØ R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

MATHFAX

```
1 REM ********************** MATHFAX *
2 DIM M$(2Ø),Q$(1)
5 M$=CHR$(125):M$(2)="          "
1Ø PRINT M$;"* MATHFAX *"
2Ø PRINT "1=add 2=subtract 3=multiply
4=divide"
25 PRINT "highest operation";:INPUT HO
:HO=HO-1
3Ø PRINT "highest number for addition"
;:INPUT HA
35 IF HO<2 THEN 5Ø
4Ø PRINT "                for multiplication"
;:INPUT HM
5Ø HN=HO:GOSUB 13ØØ:OP=R+1:PRINT M$;:O
N OP GOTO 6Ø,6Ø,9Ø,9Ø
6Ø HN=HA:GOSUB 13ØØ:N1=R:GOSUB 13ØØ:N2
=R:ON OP GOTO 7Ø,8Ø
7Ø PRINT N1;" + ";N2;:A=N1+N2:GOTO 13Ø
8Ø PRINT N1+N2;" - ";N1;:A=N2:GOTO 13Ø
9Ø HN=HM:GOSUB 13ØØ:N1=R:GOSUB 13ØØ:N2
=R:ON OP GOTO 7Ø,8Ø,1ØØ,11Ø
1ØØ PRINT N1;" * ";N2;:A=N1*N2:GOTO 13
Ø
11Ø IF N1=Ø OR N2=Ø THEN 9Ø
12Ø PRINT N1*N2;" / ";N1;:A=N2:GOTO 13
Ø
13Ø PRINT " = ";:INPUT Q:IF Q<>A THEN
16Ø
```

```
14Ø PRINT :PRINT ,,"YES"
15Ø FOR T=Ø TO 999:NEXT T:GOTO 5Ø
16Ø PRINT :PRINT ,,"NO":ON OP GOTO 17Ø
,165,18Ø,18Ø
165 NT=N1+N2:GOTO 175
17Ø NT=N2
175 PRINT ,;:GOSUB 3ØØ:PRINT ,;:NT=N1:
GOSUB 3ØØ:GOTO 15Ø
18Ø NZ=Ø
19Ø IF NZ<N1 THEN PRINT ,:NT=N2:GOSUB
3ØØ:NZ=NZ+1:GOTO 19Ø
25Ø PRINT ,,"Okay";:INPUT Q$:GOTO 5Ø
3ØØ NP=Ø
31Ø IF NP<NT THEN PRINT CHR$(Ø);:NP=NP
+1:GOTO 31Ø
32Ø PRINT :RETURN
13ØØ R=INT(LN+(HN-LN+1)*RND(1)):RETURN
```

TIMELINE

```
1 REM ********************** TIMELINE *
2 DIM C$(24),T$(39),A$(4ØØØ),K$(1)
3 DIM Y(1ØØ),Z(1ØØ)
4 OPEN #1,4,Ø,"K:"
5 C$=CHR$(125):C$(2)="          * TIMEL
INE *":PRINT C$
9 N=Ø:P=Ø
1Ø Z(N)=P:READ YR,T$:IF T$="ZZZ" THEN
3Ø
2Ø LT=LEN(T$):A$(P+1)=T$:N=N+1:Y(N)=YR
:LT=LEN(T$):P=P+LT:GOTO 1Ø
3Ø LN=N:FOR N=1 TO LN:PRINT Y(N):PRINT
 A$(Z(N-1)+1,Z(N)):PRINT :NEXT N:Y=3
4Ø PRINT C$:FOR N=Y-2 TO Y+2:PRINT Y(N
):PRINT A$(Z(N-1)+1,Z(N)):PRINT :NEXT
N
5Ø PRINT " - - - - - - - - -
- - -"
6Ø PRINT "   - earlier  + later  which
?";:GOSUB 1ØØØ:Y=Y+1
7Ø IF K$="+" THEN Y=Y+3
75 IF Y>LN-2 THEN Y=LN-2
```

```
80 IF K$="-" THEN Y=Y-5
85 IF Y<3 THEN Y=3
90 GOTO 40
100 DATA 1000,Leif Ericson discovers A
merica
110 DATA 1522,first circumnavigation o
f the earth
120 DATA 1607,Jamestown settled in Vir
ginia
130 DATA 1620,Pilgrims arrive at Plymo
uth Rock
140 DATA 1756,French & Indian War begi
ns
150 DATA 1775,James Watt invents steam
 engine
160 DATA 1776,Declaration of Independe
nce
170 DATA 1778,Captain Cook discovers H
awaii
180 DATA 1787,U.S. Constitution signed
190 DATA 1791,Bill of Rights ratified
200 DATA 1803,Louisiana Purchase
210 DATA 1807,Robert Fulton's steamboa
t up the Hudson
220 DATA 1834,Charles Babbage's analyt
ical engine
230 DATA 1846,Potato famine & U.S.-Mex
ico war
240 DATA 1849,California Gold Rush
250 DATA 1858,first trans-Atlantic cab
le
260 DATA 1861,Civil War begins
270 DATA 1865,Abraham Lincoln shot
280 DATA 1869,Golden spike: railroad a
cross U.S.
290 DATA 1888,George Eastman invents K
odak camera
999 DATA -1,ZZZ
1000 I=PEEK(764):IF I=255 THEN 1000
1010 GET #1,K:POKE 764,255
1020 K$=CHR$(K)
1090 RETURN
```

MINI-WOMBATS

```
1 REM ****************** miniWOMBATS *
2 DIM Q$(250),N$(16),J$(32),P$(32),SX$
(12),C$(1)
5 C$=CHR$(125):PRINT C$
8 POKE 702,0
9 SX=1:P$="Sienna":J$="wombat"
10 PRINT :PRINT "* Wombats !! *":LN=0
20 PRINT "1=add 2=subtract 3=multiply
4=divide"
25 PRINT "highest operation";:INPUT HO
:HO=HO-1
30 PRINT "highest number for addition"
;:INPUT HA
35 IF HO<2 THEN 50
40 PRINT "             for multiplication"
;:INPUT HM
50 PRINT C$:PRINT :PRINT :PRINT
53 PRINT "Please tell me your name";:I
NPUT N$
55 IF ASC(N$)>90 THEN GOSUB 1170:GOTO
53
60 HN=3:GOSUB 1300:IF R=3 THEN GOSUB 1
100
70 GOSUB 1300:IF R=3 THEN GOSUB 1200
80 HN=HO:GOSUB 1300:OP=R+1:IF OP>2 THE
N HN=HM:GOTO 90
85 HN=HA
90 GOSUB 1300:N1=R:GOSUB 1300:N2=R:ON
OP GOTO 100,300,500,700
100 HN=2:GOSUB 1300:ON R+1 GOTO 110,17
0,220
110 Q$=P$:GOSUB 1500:Q$(Z)=" found a b
ag containing ":GOSUB 1500
120 Q$(Z)=STR$(N1):GOSUB 1500:Q$(Z)="
":Q$(Z+1)=J$:GOSUB 1500
130 Q$(Z)="s. ":GOSUB 1400
140 GOSUB 1500:Q$(Z)="already has ":GO
```

```
SUB 1500:Q$(Z)=STR$(N2):GOSUB 1500
150 Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500:Q$
(Z)="s at home. How many does "
160 GOSUB 1420:GOSUB 1500:Q$(Z)="have
now":GOTO 290
170 Q$=P$:GOSUB 1500:Q$(Z)=" had ":GOS
UB 1500:Q$(Z)=STR$(N1):GOSUB 1500
180 Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500:Q$
(Z)="s in ":GOSUB 1480:GOSUB 1500
190 Q$(Z)="pocket, and later won ":GOS
UB 1500:Q$(Z)=STR$(N2):GOSUB 1500
200 Q$(Z)=" more in a bet with ":GOSUB
 1500:Q$(Z)=N$:GOSUB 1500
210 Q$(Z)=". How many did ":GOSUB 1420
:GOSUB 1500:Q$(Z)="have then":GOTO 290
220 Q$=P$:GOSUB 1500:Q$(Z)=" receives
2 envelopes. One contains ":GOSUB 1500
230 Q$(Z)=STR$(N1):GOSUB 1500:Q$(Z)="
":Q$(Z+1)=J$:GOSUB 1500
240 Q$(Z)="s; the other contains ":GOS
UB 1500:Q$(Z)=STR$(N2):GOSUB 1500
250 Q$(Z)=". How many does ":GOSUB 142
0:GOSUB 1500:Q$(Z)="have now":GOTO 290
290 A=N1+N2:GOTO 900
300 N2=N1+N2:HN=2:GOSUB 1300:ON R+1 GO
TO 310,340,400
310 Q$=P$:GOSUB 1500:Q$(Z)=" hides ":G
OSUB 1500:Q$(Z)=STR$(N2):GOSUB 1500
320 Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500:Q$
(Z)="s, and you find ":GOSUB 1500
330 Q$(Z)=STR$(N1):GOSUB 1500:Q$(Z)=".
 How many are still hidden":GOTO 490
340 Q$=P$:GOSUB 1500:Q$(Z)=" had too m
any ":GOSUB 1500:Q$(Z)=J$:GOSUB 1500
350 Q$(Z)="s and gave you ":GOSUB 1500
:Q$(Z)=STR$(N2):GOSUB 1500
360 Q$(Z)=". Later, ":GOSUB 1420:GOSUB
 1500:Q$(Z)="lost all of them, "
370 GOSUB 1500:Q$(Z)="and you kindly g
ave ":GOSUB 1500
```

```
380 Q$(Z)=STR$(N1):GOSUB 1500:Q$(Z)="
back. How many do you still have":GOTO
 490
400 Q$="Yesterday, ":GOSUB 1500:Q$(Z)=
P$:GOSUB 1500:Q$(Z)=" bought ":GOSUB 1
500
410 Q$(Z)=STR$(N2):GOSUB 1500:Q$(Z)="
":Q$(Z+1)=J$:GOSUB 1500
420 Q$(Z)="s, but this morning ":GOSUB
 1420:GOSUB 1500
430 Q$(Z)="could only find ":GOSUB 150
0:Q$(Z)=STR$(N1):GOSUB 1500
440 Q$(Z)=". How many were missing":GO
TO 490
490 A=N2-N1:GOTO 900
500 HN=2:GOSUB 1300:ON R+1 GOTO 510,56
0,600
510 Q$=P$:GOSUB 1500:Q$(Z)=" won ":GOS
UB 1500:Q$(Z)=STR$(N1):GOSUB 1500
520 Q$(Z)=" coupons at the fair. ":GOS
UB 1400:GOSUB 1500
530 Q$(Z)="exchanged each coupon for "
:GOSUB 1500:Q$(Z)=STR$(N2):GOSUB 1500
540 Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500:Q$
(Z)="s. How many does ":GOSUB 1420
550 GOSUB 1500:Q$(Z)="have now":GOTO 6
90
560 Q$=P$:GOSUB 1500:Q$(Z)=" built a m
achine to make ":GOSUB 1500
570 Q$(Z)=J$:GOSUB 1500:Q$(Z)="s. It h
as made ":GOSUB 1500
580 Q$(Z)=STR$(N1):GOSUB 1500:Q$(Z)="
each day for ":GOSUB 1500:Q$(Z)=STR$(N
2)
590 GOSUB 1500:Q$(Z)=" days. How many
has it made":GOTO 690
600 Q$="A flying saucer deposits ":GOS
UB 1500:Q$(Z)=STR$(N1):GOSUB 1500
610 Q$(Z)=" silvery spheres in ":GOSUB
 1500:Q$(Z)=P$:GOSUB 1500
620 Q$(Z)="'s back yard, and ":GOSUB 1
```

```
500:Q$(Z)=STR$(N2):GOSUB 1500
630 Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500:Q$
(Z)="s jumped out of each one. How man
y "
640 GOSUB 1500:Q$(Z)=J$:GOSUB 1500:Q$(
Z)="s are rampaging around ":GOSUB 150
0
650 Q$(Z)=P$:GOSUB 1500:Q$(Z)="'s hous
e right now":GOTO 690
690 A=N1*N2:GOTO 900
700 IF N1=0 THEN HN=HM:GOSUB 1300:N1=R
:GOTO 700
710 N2=N1*N2:HN=2:GOSUB 1300:ON R+1 GO
TO 720,770,820
720 Q$=P$:GOSUB 1500:Q$(Z)=" dug ":GOS
UB 1500:Q$(Z)=STR$(N1):GOSUB 1500
730 Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500:Q$
(Z)=" traps in the forest, and caught
a total of "
740 GOSUB 1500:Q$(Z)=STR$(N2):GOSUB 15
00:Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500
750 Q$(Z)="s. On the average, how many
 did ":GOSUB 1420:GOSUB 1500
760 Q$(Z)="find in each trap":GOTO 890
770 Q$=P$:GOSUB 1500:Q$(Z)=" buys a pa
cket with ":GOSUB 1500:Q$(Z)=STR$(N2)
780 GOSUB 1500:Q$(Z)=" ":Q$(Z+1)=J$:GO
SUB 1500:Q$(Z)=" seeds in it. The dire
ctions tell "
790 GOSUB 1460:GOSUB 1500:Q$(Z)="to pl
ant ":GOSUB 1500:Q$(Z)=STR$(N1):GOSUB
1500
800 Q$(Z)=" in each hole. How many hol
es should ":GOSUB 1420:GOSUB 1500
810 Q$(Z)="dig":GOTO 890
820 Q$=P$:GOSUB 1500:Q$(Z)=" shared ":
GOSUB 1500:Q$(Z)=STR$(N2):GOSUB 1500
830 Q$(Z)=" ":Q$(Z+1)=J$:GOSUB 1500:Q$
(Z)="s with ":GOSUB 1500:Q$(Z)=STR$(N1
-1)
840 GOSUB 1500:Q$(Z)=" of ":GOSUB 1480
:GOSUB 1500
```

```
850 Q$(Z)="friends. How many did each
get":GOTO 890
890 A=N2/N1
900 PRINT :PRINT " = + = + = + = + = +
 = + = + = + =":PRINT
910 LQ=LEN(Q$):L1=1:L2=36
920 IF Q$(L2,L2)=" " THEN 940
930 L2=L2-1:GOTO 920
940 PRINT Q$(L1,L2):L1=L2+1:L2=L1+34:I
F L2<LQ THEN 920
950 PRINT Q$(L1,LQ);"";
960 INPUT Q:IF Q=A THEN 1000
970 PRINT :PRINT "Woops! That's not ri
ght."
980 PRINT "The right answer is ";A;"."
990 GOTO 1090
1000 HN=3:GOSUB 1300:ON R GOTO 1010,10
20,1030
1010 PRINT "You got it!":GOTO 1090
1020 PRINT "Well done, ";N$;".":GOTO 1
090
1030 PRINT "That's right!":GOTO 1090
1090 FOR LS=1 TO 12:PRINT :NEXT LS:PRI
NT "";:GOTO 60
1100 PRINT :PRINT " ++ ++ ++ ++ ++ ++
++ ++ ++ ++ ++ ++"
1110 PRINT "Name a person";:INPUT P$
1120 IF ASC(P$)>90 THEN GOSUB 1170:GOT
O 1110
1130 PRINT "Is ";P$;" a girl or a boy"
;:INPUT SX$:SX=0:IF SX$="boy" THEN SX=
2
1140 IF SX$="girl" THEN SX=1
1150 IF SX>0 THEN RETURN
1160 PRINT "I'm sorry. I don't know an
y questions":PRINT "about ";SX$;"s. Pl
ease try again.":GOTO 1130
1170 PRINT "Person's names begin with
a CAPITAL."
1180 PRINT "Remember the SHIFT key, an
d please"
```

```
1190 PRINT "try again, ";N$;".":RETURN

1200 PRINT :PRINT " -- -- -- -- -- --
-- -- -- -- -- --"
1210 PRINT "Name an object";:INPUT J$
1290 RETURN
1300 R=INT(LN+(HN-LN+1)*RND(1)):RETURN

1400 GOSUB 1500:IF SX=1 THEN Q$(Z)="Sh
e ":RETURN
1410 Q$(Z)="He ":RETURN
1420 GOSUB 1500:IF SX=1 THEN Q$(Z)="sh
e ":RETURN
1430 Q$(Z)="he ":RETURN
1440 GOSUB 1500:IF SX=1 THEN Q$(Z)="he
rs ":RETURN
1450 Q$(Z)="his ":RETURN
1460 GOSUB 1500:IF SX=1 THEN Q$(Z)="he
r ":RETURN
1470 Q$(Z)="him ":RETURN
1480 GOSUB 1500:IF SX=1 THEN Q$(Z)="he
r ":RETURN
1490 Q$(Z)="his ":RETURN
1500 Z=LEN(Q$)+1:RETURN
```

PAINT

```
1 REM ************************ PAINT *
2 GM=3:MX=39:MY=23
3 DIM K$(1),E$(8),F$(14),S$(MX),P$(MX*
(MY+1))
4 OPEN #1,4,0,"K:"
5 GRAPHICS GM+16:X=MX/2:Y=MY/2:PC=1:CO
LOR 1
9 CF=1
10 CF=PC:GOSUB 1100:GOSUB 1000
20 IF K=45 THEN Y=Y-1:IF Y<0 THEN Y=0
30 IF K=61 THEN Y=Y+1:IF Y>MY THEN Y=M
Y
```

```
40 IF K=43 THEN X=X-1:IF X<1 THEN X=1
50 IF K=42 THEN X=X+1:IF X>MX THEN X=M
X
60 IF K=32 AND PF=1 THEN PF=0:GOTO 70
65 IF K=32 THEN PF=1
70 IF K>47 AND K<52 THEN PC=K-48:CF=PC
80 IF K$="Q" THEN 900
90 IF K$="Z" THEN 800
100 IF K$="S" THEN 300
110 IF K$="L" THEN 500
120 IF K$="C" THEN 200
190 GOTO 10
200 CV=0:PV=PC-1:IF PV<0 THEN PV=4
210 SETCOLOR PV,CV,6:GOSUB 1000:IF K=1
55 THEN 250
220 IF K$="+" THEN CV=CV+1:IF CV>15 TH
EN CV=15
230 IF K$="-" THEN CV=CV-1:IF CV<0 THE
N CV=0
240 GOTO 210
250 LV=0
260 SETCOLOR PV,CV,LV:GOSUB 1000:IF K=
155 THEN 10
270 IF K$="+" THEN LV=LV+2:IF LV>14 TH
EN LV=14
280 IF K$="-" THEN LV=LV-2:IF LV<0 THE
N LV=0
290 GOTO 260
300 GOSUB 400
310 PRINT "FILE TO SAVE";:INPUT E$
320 F$="D:":F$(3)=E$:F$(LEN(F$)+1)=".P
IX"
330 OPEN #2,8,0,F$
340 FOR SY=0 TO MY:PP=1+MX*SY:S$=P$(PP
,PP+38):PRINT #2;S$:NEXT SY
350 PRINT #2;"ZZZ":CLOSE #2
390 LX=MX:LY=MY:GOSUB 600:GOTO 10
400 PP=1:FOR SY=0 TO MY:FOR SX=1 TO MX
410 LOCATE SX,SY,P:P$(PP)=STR$(P):PP=P
P+1
```

```
420 NEXT SX
430 PLOT 1,SY:COLOR 1:DRAWTO 1,SY
440 NEXT SY
490 RETURN
500 PRINT "FILE TO LOAD";:INPUT E$
510 F$="D:":F$(3)=E$:F$(LEN(F$)+1)=".P
IX"
520 OPEN #2,4,0,F$:LY=0:PP=1
540 INPUT #2;S$:IF S$="ZZZ" THEN 580
550 P$(PP)=S$:LX=LEN(S$):PP=PP+LX
560 LY=LY+1:GOTO 540
580 CLOSE #2:LY=LY-1
590 GOSUB 600:GOTO 10
600 GRAPHICS GM+16:PP=1:FOR SY=0 TO LY
:FOR SX=1 TO LX
610 CF=VAL(P$(PP,PP)):COLOR CF
620 PLOT SX,SY:DRAWTO SX,SY:PP=PP+1
630 NEXT SX:NEXT SY
690 RETURN
800 GOSUB 400:PRINT "ZAP IT";
810 INPUT K$:IF K$="N" THEN LX=MX:LY=M
Y:GOSUB 600:GOTO 10
890 GRAPHICS GM+16:GOTO 10
900 GOSUB 400:PRINT "QUIT":INPUT K$:IF
 K$="N" THEN LX=MX:LY=MY:GOSUB 600:GOT
O 10
990 END
1000 I=PEEK(764):IF I=255 THEN 1000
1010 GET #1,K:POKE 764,255
1030 IF PF=0 THEN CF=OC
1040 IF PF=1 THEN CF=PC
1050 GOSUB 1100
1080 K$=CHR$(K)
1090 RETURN
1100 LOCATE X,Y,OC:COLOR CF:PLOT X,Y:D
RAWTO X,Y
1190 RETURN
```

BARS

```
1 REM *************************** BARS *
2 DIM C$(40),B$(7),T$(38),S$(38),A$(26
6)
3 DIM L(7),P(7)
5 C$=CHR$(125):C$(2)="         * BARS *"
:PRINT C$
10 READ B:IF B=-1 THEN 30
20 B$(LEN(B$)+1)=CHR$(B):GOTO 10
30 READ LN
40 READ HN
50 READ NB
60 READ T$:LT=LEN(T$):M=(38-LT)/2:C$(1
+M)=T$
70 P=0:N=0
80 READ S$,V:L(N)=V:LS=LEN(S$):P(N)=P:
A$(P+1)=S$:P(N)=P:P=P+LS:N=N+1:IF N<NB
 THEN 80
90 P(N)=P
100 POKE 752,1:PRINT C$:PRINT
110 I=(HN-LN)/5:IB=36/(HN-LN)
120 FOR N=0 TO 5:PRINT INT(LN+(N*I));"
     ";:NEXT N:PRINT
130 PRINT :FOR BN=0 TO NB-1
140 FOR L=1 TO L(BN)*IB:PRINT B$(BN+1,
BN+1);:NEXT L:PRINT " ";L(BN)
150 PRINT A$(P(BN)+1,P(BN+1)):PRINT :N
EXT BN
200 GOTO 200
900 DATA 0,8,10,19,79,88,123,-1
910 DATA 0,75,6
920 DATA LIFE EXPECTANCY - MALES
930 DATA USA,69
940 DATA COLOMBIA,44
```

```
950 DATA NORWAY,71
960 DATA JAPAN,71
970 DATA INDIA,42
980 DATA AUSTRALIA,68
```

```
Ø
   15
      3Ø
         45
            6Ø
               75
```

```
1 REM ************************* BARS *
2 DIM C$(4Ø),B$(7),T$(38),S$(38),A$(26
6),Q$(1),P$(7)
3 DIM L(7),P(7)
5 C$=CHR$(125):C$(2)="          * BARS *"
:PRINT C$
1Ø READ B:IF B=-1 THEN 3Ø
2Ø B$(LEN(B$)+1)=CHR$(B):GOTO 1Ø
3Ø READ LN
4Ø READ HN
5Ø READ NB
6Ø READ T$:LT=LEN(T$):M=(38-LT)/2:C$(1
+M)=T$
7Ø P=Ø:N=Ø
8Ø READ S$,V:L(N)=V:LS=LEN(S$):P(N)=P:
A$(P+1)=S$:P(N)=P:P=P+LS:N=N+1:IF N<NB
 THEN 8Ø
9Ø P(N)=P
1ØØ POKE 752,1:PRINT C$:PRINT
11Ø I=(HN-LN)/5:IB=36/(HN-LN)
12Ø FOR N=Ø TO 5:PRINT INT(LN+(N*I));"
     ";:NEXT N:PRINT
```

```
130 PRINT :FOR BN=Ø TO NB-1
140 FOR L=1 TO L(BN)*IB:PRINT B$(BN+1,
BN+1);:NEXT L:PRINT " ";L(BN)
150 PRINT A$(P(BN)+1,P(BN+1)):PRINT :N
EXT BN

2ØØ PRINT "PRINT IT";:INPUT Q$:N=Ø:LPR
INT T$:LPRINT
21Ø READ P:IF P=-1 THEN 23Ø
22Ø P$(N+1)=CHR$(P):N=N+1:GOTO 21Ø
23Ø T$="":FOR N=Ø TO 5:T$(LEN(T$)+1)=S
TR$(INT(LN+(N*I))):T$(LEN(T$)+1)="
 ":NEXT N:LPRINT T$
24Ø LPRINT :FOR BN=Ø TO NB-1:T$=""
25Ø FOR L=1 TO L(BN)*IB:T$(LEN(T$)+1)=
P$(BN+1,BN+1):NEXT L:T$(LEN(T$)+1)=" "
:T$(LEN(T$)+1)=STR$(L(BN)):LPRINT T$
26Ø LPRINT A$(P(BN)+1,P(BN+1)):LPRINT
:NEXT BN
9ØØ DATA Ø,8,1Ø,19,79,88,123,-1
91Ø DATA Ø,75,6
92Ø DATA LIFE EXPECTANCY - MALES
93Ø DATA USA,69
94Ø DATA COLOMBIA,44
95Ø DATA NORWAY,71
96Ø DATA JAPAN,71
97Ø DATA INDIA,42
98Ø DATA AUSTRALIA,68
99Ø DATA 56,35,42,72,159,88,78,-1
```

SORT

```
1 REM *************************** SORT *
2 DIM A$(4000),D$(4000),I$(1),B$(40),C
$(40),M$(40),Q$(1),T$(40)
3 DIM L(100),M(100),S(100)
4 OPEN #1,4,0,"K:"
6 PRINT CHR$(125):N=1:LD=40
8 FOR N=1 TO LD:M$(LEN(M$)+1)=" ":NEXT
 N:N=1
10 PRINT ,"SORT"
11 PRINT "1: ENTER DATA"
12 PRINT "2: SORT"
13 PRINT "3: DISPLAY"
19 PRINT "9: QUIT"
20 PRINT ,"OPTION";:INPUT Q
30 ON Q GOTO 100,200,300,10,10,10,10,1
0,900
40 GOTO 10
100 PRINT "ENTER DATA         ZZZ TO Q
UIT"
110 PRINT N;":";:GOSUB 1000:PRINT
120 IF T$="ZZZ" THEN ND=N-1:GOTO 10
130 A$(1+LD*(N-1))=T$:L(N)=LEN(T$)-1
140 N=N+1:GOTO 110
200 PRINT "SORTING":FOR N=1 TO ND:S(N)
=1:NEXT N:PL=1
210 N=1
220 IF S(N)=0 THEN N=N+1:GOTO 220
225 P=N+1:LN=N:SP=1+LD*(N-1):B$=A$(SP,
SP+L(N))
230 IF S(P)=0 THEN 260
240 SQ=1+LD*(P-1):C$=A$(SQ,SQ+L(P)):PR
INT B$,C$,:IF B$<C$ THEN 260
250 T$=B$:B$=C$:C$=T$:LN=P:PRINT "SWAP
PING";
260 PRINT :P=P+1:IF P<=ND THEN 230
```

```
270 SY=1+LD*(PL-1):D$(SY)=B$:M(PL)=LEN
(B$)-1:PL=PL+1:S(LN)=0:IF PL=ND THEN 2
80
275 GOTO 210
280 SY=1+LD*(ND-1):D$(SY)=C$:M(ND)=LEN
(C$)-1
290 PRINT " = = = DONE = = =":A$=D$:FO
R N=1 TO ND:L(N)=M(N):NEXT N
300 FOR N=1 TO ND
310 SP=1+LD*(N-1):PRINT N;"  ";A$(SP,S
P+L(N))
320 NEXT N
390 PRINT ,,"MENU";:INPUT Q$:GOTO 10
900 PRINT "DONE":END
1000 T$=""
1010 I=PEEK(764):IF I=255 THEN 1010
1020 GET #1,K:POKE 764,255
1030 IF K=155 THEN RETURN
1070 I$=CHR$(K):PRINT I$;
1080 T$(LEN(T$)+1)=I$
1090 GOTO 1010
```

SUPERSORT

```
1 REM ************************SSORT *
2 DIM A$(4000),D$(4000),I$(1),B$(40),C
$(40),M$(40),Q$(1),T$(40),E$(8),F$(14)
,CL$(1)
3 DIM L(100),M(100),S(100)
4 OPEN #1,4,0,"K:"
6 CL$=CHR$(125):PRINT CL$
9 N=1:LD=40:GOTO 800
10 PRINT ,"     SORT"
11 PRINT "1: ENTER DATA"
12 PRINT "2: SORT          6: PRINT"
13 PRINT "3: DISPLAY       7: EDIT"
14 PRINT "4: SAVE          8: CLEAR"
15 PRINT "5: LOAD          9: QUIT"
20 PRINT ,,"OPTION";:INPUT Q
30 ON Q GOTO 100,200,300,400,500,600,7
```

```
ØØ,8ØØ,9ØØ
4Ø GOTO 1Ø
1ØØ PRINT "ENTER DATA           ZZZ TO Q
UIT"
11Ø PRINT N;":";:GOSUB 1ØØØ:PRINT
12Ø IF T$="ZZZ" THEN ND=N-1:GOTO 1Ø
13Ø A$(1+LD*(N-1))=T$:L(N)=LEN(T$)-1
14Ø N=N+1:GOTO 11Ø
2ØØ PRINT "SORTING":FOR N=1 TO ND:S(N)
=1:NEXT N:PL=1
21Ø N=1
22Ø IF S(N)=Ø THEN N=N+1:GOTO 22Ø
225 P=N+1:LN=N:SP=1+LD*(N-1):B$=A$(SP,
SP+L(N))
23Ø IF S(P)=Ø THEN 26Ø
24Ø SQ=1+LD*(P-1):C$=A$(SQ,SQ+L(P)):PR
INT B$,C$,:IF B$<C$ THEN 26Ø
25Ø T$=B$:B$=C$:C$=T$:LN=P:PRINT "SWAP
PING";
26Ø PRINT :P=P+1:IF P<=ND THEN 23Ø
27Ø SY=1+LD*(PL-1):D$(SY)=B$:M(PL)=LEN
(B$)-1:PL=PL+1:S(LN)=Ø:IF PL=ND THEN 2
8Ø
275 GOTO 21Ø
28Ø SY=1+LD*(ND-1):D$(SY)=C$:M(ND)=LEN
(C$)-1
29Ø PRINT " = = = DONE = = =":A$=D$:FO
R N=1 TO ND:L(N)=M(N):NEXT N
3ØØ FOR N=1 TO ND
31Ø SP=1+LD*(N-1):PRINT N;"  ";A$(SP,S
P+L(N))
32Ø NEXT N
39Ø PRINT ,,"MENU";:INPUT Q$:GOTO 1Ø
4ØØ PRINT ,"FILENAME FOR SAVE";:INPUT
E$
41Ø F$="D:":F$(3)=E$:F$(LEN(F$)+1)=".D
TA":OPEN #2,8,Ø,F$
42Ø FOR N=1 TO ND
43Ø SP=1+LD*(N-1):T$=A$(SP,SP+L(N)):PR
INT #2;T$:PRINT ">";T$
```

```
440 NEXT N:PRINT #2;"ZZZ":CLOSE #2
490 GOTO 10
500 PRINT ,"FILENAME TO LOAD";:INPUT E
$
510 F$="D:":F$(3)=E$:F$(LEN(F$)+1)=".D
TA":OPEN #2,4,0,F$
520 INPUT #2,T$:IF T$="ZZZ" THEN 590
530 SP=1+LD*(N-1):A$(SP)=T$:L(N)=LEN(T
$)-1
535 PRINT "<";T$
540 N=N+1:GOTO 520
590 ND=N-1:CLOSE #2:GOTO 10
600 PRINT "PRINT LINE NUMBERS (Y/N)";:
INPUT Q$
610 FOR N=1 TO ND
620 IF Q$<>"Y" THEN 640
630 SP=1+LD*(N-1):LPRINT N;"   ";A$(SP,
SP+L(N)):GOTO 650
640 SP=1+LD*(N-1):LPRINT A$(SP,SP+L(N)
)
650 NEXT N
690 GOTO 10
700 PRINT ,"LINE TO CORRECT";:INPUT X:
IF X=0 THEN 10
710 SP=1+LD*(X-1):PRINT X;"  ";A$(SP,SP
+L(X))
720 PRINT "   ";:GOSUB 1000:PRINT
730 IF T$="ZZZ" THEN 10
740 IF T$="" THEN 700
750 D$="":IF X<ND THEN D$=A$(SP+LD,LEN
(A$))
760 A$(SP)=T$:L(X)=LEN(T$)-1:A$(SP+LD)
=D$
790 GOTO 700
800 PRINT "CLEARING MY MIND ..."
810 FOR Z=1 TO 1000:A$(Z)=" ":NEXT Z
820 FOR Z=1 TO 100:S(Z)=0:N=1
890 PRINT CL$:GOTO 10
900 PRINT "DONE":END
1000 T$=""
1010 I=PEEK(764):IF I=255 THEN 1010
```

```
1020 GET #1,K:POKE 764,255
1030 IF K=155 THEN RETURN
1070 I$=CHR$(K):PRINT I$;
1080 T$(LEN(T$)+1)=I$
1090 GOTO 1010
```

ANYBASE

```
1 REM * ANYBASE *
2 DIM Q$(9),C$(9),D$(9),N(9),M(9)
3 FOR N=0 TO 9:N(N)=0:M(N)=0:NEXT N
10 PRINT "      NUMBER";:INPUT Q$
20 PRINT "    ITS BASE";:INPUT B2
30 PRINT "TARGET BASE";:INPUT B1
70 PRINT "";:GOSUB 300:GOSUB 200
90 IF C$=Q$ THEN PRINT "":RUN
100 GOTO 70
200 N=0
210 N(N)=N(N)+1:IF N(N)<B1 THEN 230
220 N(N)=0:N=N+1:GOTO 210
230 IF N>HN THEN HN=N
240 FOR PP=1 TO 16:PRINT "";:NEXT PP:P
RINT ">";:FOR D=HN TO 0 STEP -1
250 IF N(D)<10 THEN D$=STR$(N(D)):GOTO
 270
260 D$=CHR$(55+N(D))
270 PRINT D$;:NEXT D
290 PRINT :RETURN
300 M=0:C$=""
310 M(M)=M(M)+1:IF M(M)<B2 THEN 330
320 M(M)=0:M=M+1:GOTO 310
330 IF M>HM THEN HM=M
340 FOR PP=1 TO 16:PRINT "";:NEXT PP:P
RINT ">";:FOR D=HM TO 0 STEP -1
350 IF M(D)<10 THEN D$=STR$(M(D)):GOTO
 370
360 D$=CHR$(55+M(D))
370 PRINT D$;:C$(LEN(C$)+1)=D$:NEXT D
390 PRINT :RETURN
490 B1=16
500 GOSUB 200:GOTO 500
```

CALC

```
1 REM * CALC *
2 DIM A$(100),I$(1),V$(100),O(100),V(1
00)
10 MO=0:PRINT "CALC";:INPUT A$
20 LA=LEN(A$):N=1:P=1
30 I$=A$(P,P)
40 IF I$="+" THEN O(N)=1:GOTO 180
50 IF I$="-" THEN O(N)=2:GOTO 180
60 IF I$="*" THEN O(N)=3:GOTO 180
70 IF I$="/" THEN O(N)=4:GOTO 180
80 IF I$>"/" AND I$<":" THEN 170
90 IF I$="." THEN 170
100 IF I$="A" THEN 150
110 IF I$="=" THEN 130
120 GOTO 190
130 MO=1:OA=VAL(A$(P+1)):GOTO 200
150 V$=STR$(OA):GOTO 190
170 V$(LEN(V$)+1)=I$:GOTO 190
180 V(N)=VAL(V$):V$="":N=N+1
190 P=P+1:IF P<=LA THEN 30
200 V(N)=VAL(V$):V$="":LN=N:N=1
210 IF LN=1 THEN 400
220 IF O(N)=1 THEN V(N)=V(N)+V(N+1):GO
SUB 300:N=1:GOTO 210
230 IF O(N)=2 THEN V(N)=V(N)-V(N+1):GO
SUB 300:N=1:GOTO 210
240 N=N+1:IF N<LN THEN 210
250 N=1:IF LN=1 THEN 400
260 IF O(N)=3 THEN V(N)=V(N)*V(N+1):GO
SUB 300:N=1:GOTO 250
270 IF O(N)=4 THEN V(N)=V(N)/V(N+1):GO
SUB 300:N=1:GOTO 250
280 N=N+1:IF N<LN THEN 250
290 N=1:IF LN=1 THEN 400
300 FOR M=N+1 TO LN-1
```

```
310 V(M)=V(M+1):O(M-1)=O(M)
320 NEXT M:LN=LN-1:RETURN
400 PRINT "";:FOR P=1 TO LA+6:PRINT ""
;:NEXT P
410 IF MO=1 THEN 500
420 OA=V(1):PRINT "=";OA:GOTO 10
500 IF QA=V(1) THEN PRINT "YES":GOTO 1
0
510 PRINT "NO":GOTO 10
```

```
1 REM ********************** DATEBOOK *
2 DIM A$(4000),D$(4000),I$(1),B$(76),C
$(76),M$(76),Q$(1),T$(76),E$(76),F$(76
),N$(76),CL$(1)
3 DIM L(100,2),M(100,2),S(100)
4 OPEN #1,4,0,"K:"
6 CL$=CHR$(125):PRINT CL$
9 N=1:LD=76:L1=14:L2=39:REM GOTO 800
10 PRINT ,"   DATEBOOK"
11 PRINT "1: ENTER DATA"
12 PRINT "2: SORT           6: PRINT"
13 PRINT "3: DISPLAY        7: EDIT"
14 PRINT "4: SAVE           8: CLEAR"
15 PRINT "5: LOAD           9: QUIT"
20 PRINT ,,"OPTION";:INPUT Q
30 ON Q GOTO 100,200,300,400,500,600,7
00,800,900
40 GOTO 10
100 PRINT "ENTER DATA          ZZZ TO Q
UIT"
110 PRINT "DATE/TIME:";:GOSUB 1000:PRI
NT
120 IF T$="ZZZ" THEN ND=N-1:GOTO 10
130 D$=T$:PRINT "     EVENT:";:GOSUB 10
00:PRINT
140 E$=T$:PRINT "      NOTE:";:GOSUB 10
00:PRINT
```

```
150 N$=T$:GOSUB 1100
180 A$(1+LD*(N-1))=T$
190 N=N+1:GOTO 110
200 PRINT "SORTING":FOR N=1 TO ND:S(N)
=1:NEXT N:FL=1
210 N=1
220 IF S(N)=0 THEN N=N+1:GOTO 220
225 P=N+1:LN=N:SP=1+LD*(N-1):B$=A$(SP,
SP+L2+L(N,2)-1)
230 IF S(P)=0 THEN 260
240 HN=P:SQ=1+LD*(P-1):C$=A$(SQ,SQ+L2+
L(P,2)-1):IF B$<C$ THEN 260
250 T$=B$:B$=C$:C$=T$:LN=P:HN=N
260 P=P+1:IF P<=ND THEN 230
270 SY=1+LD*(FL-1):D$(SY)=B$:FOR ZL=0
TO 2:M(FL,ZL)=L(LN,ZL):NEXT ZL:FL=FL+1
:S(LN)=0:IF FL=ND THEN 280
275 GOTO 210
280 SY=1+LD*(ND-1):D$(SY)=C$:FOR ZL=0
TO 2:M(ND,ZL)=L(HN,ZL):NEXT ZL
290 PRINT " = = = DONE = = =":A$=D$
295 FOR N=1 TO ND:FOR ZL=0 TO 2:L(N,ZL
)=M(N,ZL):NEXT ZL:NEXT N:GOTO 310
300 GOSUB 1400:LM=LEN(M$)
310 FOR N=1 TO ND:SP=1+LD*(N-1):T$=A$(
SP,SP+L2+L(N,2)-1)
320 GOSUB 1150:IF LM=0 THEN 370
330 ON FM GOTO 340,350,360
340 IF LM>LEN(D$) THEN 380
343 IF M$=D$(1,LM) THEN 370
346 GOTO 380
350 IF LM>LEN(E$) THEN 380
353 IF M$=E$(1,LM) THEN 370
356 GOTO 380
360 IF LM>LEN(N$) THEN 380
363 IF M$=N$(1,LM) THEN 370
366 GOTO 380
370 PRINT D$,E$:PRINT "   ";N$
380 NEXT N
390 PRINT ,,"MENU";:INPUT Q$:GOTO 10
```

```
400 PRINT ,"FILENAME FOR SAVE";:INPUT
E$
410 F$="D:":F$(3)=E$:F$(LEN(F$)+1)=".D
TA":OPEN #2,8,0,F$
420 FOR N=1 TO ND
430 SP=1+LD*(N-1):T$=A$(SP,SP+L2+L(N,2
)-1):GOSUB 1150:IF D$="" THEN 480
440 PRINT #2;D$:PRINT #2;E$:PRINT #2;N
$:PRINT ">";D$,E$
480 NEXT N:PRINT #2;"ZZZ"
490 CLOSE #2:GOTO 10
500 PRINT ,"FILENAME TO LOAD";:INPUT E
$
510 F$="D:":F$(3)=E$:F$(LEN(F$)+1)=".D
TA":OPEN #2,4,0,F$
520 INPUT #2,D$:IF D$="ZZZ" THEN 590
530 INPUT #2,E$:INPUT #2,N$
540 GOSUB 1100:SP=1+LD*(N-1):A$(SP)=T$
545 PRINT "<";D$,E$
550 N=N+1:GOTO 520
590 ND=N-1:CLOSE #2:GOTO 10
600 GOSUB 1400:LM=LEN(M$)
610 FOR N=1 TO ND:SP=1+LD*(N-1):T$=A$(
SP,SP+L2+L(N,2)-1)
620 GOSUB 1150:IF LM=0 THEN 670
630 ON FM GOTO 640,650,660
640 IF LM>LEN(D$) THEN 680
643 IF M$=D$(1,LM) THEN 670
646 GOTO 680
650 IF LM>LEN(E$) THEN 680
653 IF M$=E$(1,LM) THEN 670
656 GOTO 680
660 IF LM>LEN(N$) THEN 680
663 IF M$=N$(1,LM) THEN 670
666 GOTO 680
670 LPRINT D$,E$,N$
680 NEXT N
690 GOTO 10
700 GOSUB 1400:LM=LEN(M$):IF FM=0 THEN
 10
```

```
710 FOR N=1 TO ND:SP=1+LD*(N-1):T$=A$(
SP,SP+L2+L(N,2)-1)
720 GOSUB 1150:IF LM=0 THEN 770
730 ON FM GOTO 740,750,760
740 IF LM>LEN(D$) THEN 790
743 IF M$=D$(1,LM) THEN 770
746 GOTO 790
750 IF LM>LEN(E$) THEN 790
753 IF M$=E$(1,LM) THEN 770
756 GOTO 790
760 IF LM>LEN(N$) THEN 790
763 IF M$=N$(1,LM) THEN 770
766 GOTO 790
770 GOSUB 1700:IF T$="ZZZ" THEN N=ND+1
:GOTO 10
780 D$="":IF N=ND THEN 785
783 D$=A$(SP+LD,LEN(A$))
785 A$(SP)=T$:A$(SP+LD)=D$
790 NEXT N:GOTO 700
800 PRINT "CLEARING MY MIND ..."
810 FOR Z=1 TO 1000:A$(Z)=" ":NEXT Z
820 FOR Z=1 TO 100:S(Z)=0:NEXT Z
890 N=1:PRINT CL$:GOTO 10
900 PRINT "DONE":END
1000 T$=""
1010 I=PEEK(764):IF I=255 THEN 1010
1020 GET #1,K:POKE 764,255
1030 IF K=155 THEN RETURN
1070 I$=CHR$(K):PRINT I$;
1080 T$(LEN(T$)+1)=I$
1090 GOTO 1010
1100 T$=D$:T$(L1)=E$:T$(L2)=N$
1110 L(N,0)=LEN(D$)-1:L(N,1)=LEN(E$)-1
:L(N,2)=LEN(N$)-1
1140 RETURN
1150 D$="":IF L(N,0)<0 THEN 1160
1155 D$=T$(1,1+L(N,0))
1160 E$="":IF L(N,1)<0 THEN 1170
1165 E$=T$(L1,L1+L(N,1))
1170 N$="":IF L(N,2)<0 THEN 1190
```

```
1180 N$=T$(L2,L2+L(N,2))
1190 RETURN
1400 M$="":PRINT "FIELD TO MATCH";
1410 INPUT F$:IF F$="" THEN FM=0:GOTO
1490
1420 FM=VAL(F$):PRINT " DATA TO MATCH"
;
1430 INPUT M$
1490 RETURN
1700 PRINT D$,E$:PRINT "   ";N$
1710 PRINT "DATE/TIME:";:GOSUB 1000:PR
INT
1720 IF T$="ZZZ" THEN RETURN
1740 D$=T$
1750 PRINT "     EVENT:";:GOSUB 1000:PR
INT :IF T$="" THEN 1770
1760 E$=T$
1770 PRINT "      NOTE:";:GOSUB 1000:PR
INT :IF T$="" THEN 1790
1780 N$=T$
1790 GOSUB 1100:RETURN
```

$6.95

HOMEWORK HELPER FOR THE ATARI®

Herbert Kohl

Most school-age children agree on at least one thing—homework is not fun! The very mention of this nightly ritual in most homes elicits moans and groans along with some pretty creative excuses!

Michael Potts knows the problems associated with homework and has developed **HOMEWORK HELPER,** an exciting new group of programs that takes the anxiety out of homework and replaces it with logical, workable alternatives!

This important addition to the Creative Pastimes series includes programs to help your child with virtually all major coursework. Spelling, vocabulary, history, and math programs can be tailored to suit your child's individual study needs. In addition, valuable sections on developing good study habits, organizing school assignments, and important note-keeping programs maximize study time.

Let your child benefit from a winning combination—**HOMEWORK HELPER** and your Atari computer!

A Creative Pastimes Book
RESTON PUBLISHING COMPANY, INC.
A Prentice-Hall Company
Reston, Virginia

0 21898 28598 0

0-8359-2859-4